A. BELLIDO

SPECTRUM

PLUS ULTRA

La enciclopedia del Spectrum



TOMO

1

PARANINFO S.A.

SPECTRUM
PLUS ULTRA

La enciclopedia del SPECTRUM 1

A. BELLIDO

SPECTRUM

PLUS ULTRA

La enciclopedia del SPECTRUM 1

PARANINFO S.A.
1985
MADRID

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 84-283-1392-X (Obra completa)
ISBN: 84-283-1391-1 (Tomo 1)
Depósito Legal: M-16.615-1985 (Tomo I)
Depósito Legal: M-16.617-1985 (Obra completa)



Magallanes, 25 - 28015 - MADRID (3-3510)

Indice general

Spectrum: PLUS ULTRA
Tomo I

Indice de materias

SECCION I

SECCION II

SECCION III

APENDICES

A...

Zudia y Antonio,
sin cuya ayuda
habría terminado
antes.
Tal vez

El autor

Prólogo

En los últimos años hemos asistido a un vertiginoso desarrollo de la informática y muy especialmente de la microinformática.

Los ordenadores personales SINCLAIR han sido un claro ejemplo y sus ventas se cuentan por millones en todo el mundo. El éxito del Ordenador Personal Sinclair ZX Spectrum ha sido motivado no sólo por lo original de su concepción sino más bien por la enorme cantidad de literatura y software que sobre él se ha publicado.

Los libros que el autor ha publicado hasta la aparición de éste han corrido un camino paralelo al del Spectrum y en lo que al mercado de lengua española se refiere han sido el complemento al manual del usuario, cuando no el propio manual.

La literatura sobre SPECTRUM es enorme, pero en castellano y pensada para el lector en castellano, era inexistente hasta la aparición de los libros del AUTOR.

En la mayoría de los casos los manuales de usuarios han sido escritos por expertos que poco o nada han tenido en cuenta las características de sus futuros lectores.

Los libros escritos por Antonio Bellido son todo lo contrario, esto es, han sido pensados por un usuario y para el usuario. La claridad en la presentación, la sencillez en la exposición y los ejemplos que se acompañan vienen a complementar, cuando no a aclarar, los conceptos en los que el lector trata de profundizar.

Este nuevo libro en relación al Ordenador Personal SINCLAIR SPECTRUM no dudamos acompañará en éxito a los anteriores del Autor y harán que el SPECTRUM PLUS llegue en efecto "PLUS ULTRA".

RICARDO GARCIA GETE
Director Comercial de la
División de Microelectrónica
de INVESTRONICA

Introducción

En la presente publicación resumo lo mejor –y más adecuado a los propósitos didácticos que persigo– de otros títulos mios y añado temas no tocados por mí hasta el momento.

Sugiero al posible lector de este libro que revise el INDICE de materias, ya que la SECCION 1ª y la 3ᵉ están extraídas de "Cómo programar su Spectrum" y de "Los colores y los gráficos en el Spectrum".

El contenido de los dos volúmenes en que se divide este trabajo es igualmente útil para los usuarios de cualquier tipo de Spectrum, no obstante existen pequeñas diferencias en el manejo del teclado entre el *plus* y el modelo convencional y, aunque en los respectivos manuales está claramente explicado todo lo referente a este punto, en el APENDICE A se estudia con detalle ambas posibilidades.

Aquellas personas que ya estén familiarizadas con la programación en BASIC, pueden pasar directamente a la SECCION 2ª, donde encontrarán las fichas explicativas de todas las *palabras* que utiliza el lenguaje BASIC de Sinclair, excepción hecha de las destinadas a la confección de gráficos y el color que se estudian en la SECCION 3ª.

El tomo primero finaliza con tres apéndices dedicados a los diagramas de flujo, las funciones lógicas aplicadas y el troceado de cadenas respectivamente.

En el tomo segundo nos encontramos en primer lugar con la SECCION 4ª, donde hayamos caminos que conducen a la protección (o desprotección) de programas, líneas y, en definitiva, se suministran ciertas tretas que, normalmente, no están al alcance de los usuarios. De esta sección sale el libro "Los trucos del Spectrum" editado también por Paraninfo.

A continuación, la SECCION 5ª trata el código máquina de forma que el lector lo vea en su verdadera dimensión, proveyéndole de los conocimientos suficientes que le permitan decidir si quiere –o no– profundizar en este terreno. Esta sección es un resumen del libro "Curso de Iniciación al Código Máquina", también editado por Paraninfo con ensamblador opcional en casete.

La SECCION 6ª pone fin al Segundo tomo con una amplia introducción al uso del microdrive.

Un detallado apéndice dedicado a los sistemas de numeración, un plano de la memoria minuciosamente explicado y un croquis de acceso rápido a los diferentes comandos del teclado completan la obra.

Espero que le sea de utilidad.

El autor

Sección I

Cómo programar el SPECTRUM

INTRODUCCION PARA PRINCIPIANTES

En esta primera parte he asumido que el lector está dando su primer paso dentro del mundo de los ordenadores y, específicamente, en una parcela tan particular como es la programación en BASIC. También supongo unos conocimientos nulos de la lengua inglesa. Si usted domina el inglés, todo le resultará más fácil; si además ya tiene conocimientos de programación, quizá deba dirigirse directamente a la SECCION II.

Si usted no está familiarizado con el teclado de su Spectrum le recomiendo que lea el APENDICE A..

EL BASIC COMO UN IDIOMA

1-1

No es lo mismo "Cervantes escribió el Quijote" que "Quijote escribió el Cervantes". Es claro que el idioma español tiene unas reglas, gracias a las cuales, todos los hispanoparlantes nos podemos comunicar.

Si quisiéramos entendernos con los suecos, habría que saber sueco o, muy probablemente, inglés, dado que la mayoría de los suecos entienden este idioma.

Pues bien, algo similar nos sucede con los ordenadores.

Los humanos, para entenderse con los ordenadores, han desarrollado una serie de lenguajes, denominados de alto nivel, tales como el FORTH, COBOL o BASIC. Todos tienen sus propias reglas sintácticas, las cuales son mucho más estrictas que las que exige un idioma hablado o escrito entre personas.

Cada una de las palabras que integran estos lenguajes de alto nivel no pueden ser directamente entendidas por el microprocesador (o "cerebro") del microordenador —un microprocesador sólo puede ejecutar microinstrucciones de su muy particular idioma, cuyo nombre es "código máquina"—, consiguientemente, una palabra de cualquier idioma de alto nivel está formada por varias microinstrucciones concatenadas de forma tal que, finalmente, realicen lo que el programador espera.

Todos estos idiomas han sido desarrollados por angloparlantes y, consiguientemente, en inglés. Esta es nuestra dificultad adicional.

A partir de ahora se hará especial hincapié en la forma en que nuestro cerebro debe estructurarse para dirigirse al computador, y en recoger el concepto que cada palabra inglesa del lenguaje BASIC tiene en español.

Para conseguir estos dos últimos objetivos haremos del Spectrum nuestro interlocutor y profesor.

Su Spectrum viene acompañado por una INTRODUCCION a este ordenador, un MANUAL de programación, una cinta casete de DEMOSTRACION, un transformador, un cable de conexión al televisor y un cable de conexión a un casete. Con esto, un TV y un casete usted está en condiciones de seguir este curso.

El esquema de conexiones (Fig. 1) y la forma de poner en marcha su Spectrum está perfectamente explicado en la INTRODUCCION citada en el párrafo anterior y, por tanto, no se comenta aquí. Solamente indicarle que el Spectrum emitirá un ligero zumbido al conectarlo que no debe preocuparle.

Para las personas no habituadas a este tipo de máquinas sumarizamos, como apoyo a la cinta de demostración, la función que cumplen todos los artilugios que conforman su sistema de computación y que, básicamente, son los mismos en los ordenadores de mayor entidad:

***Caja del Spectrum.**— Contiene toda la electrónica para el manejo y almacenamiento de la información (16K o 48K) y posee la adecuada conexión interna al teclado y externa al casete (EAR) para permitir la entrada de datos al computador más dos conexiones externas (TV y MIC) para permitir la salida de información hacia el televisor y el casete. La ranura de mayor dimensión que se contempla en la parte posterior de la caja está destinada a permitir el acoplamiento de diferentes periféricos, tales como la impresora y la caja de expansiones.

***Teclado.**— Este dispositivo es sólo de entrada, ya que, por medio del tecleado adecuado, se consigue introducir información al ordenador.

***Transformador.**— Adecua la energía eléctrica a las necesidades de trabajo de la máquina (9Vs. y 1.2As).

***Televisor.**— Este dispositivo es sólo de salida, ya que, a través de él, el ordenador emite sus respuestas e información en general, permitiendo una interactividad total entre el usuario y su computador.

***Casete.**— El magnetofón es un medio de almacenamiento externo —ajeno a la electrónica interna del computador— que permite transferir los datos y programas desde el ordenador a una cinta magnética (SAVE) y en sentido contrario (LOAD). Los ordenadores más costosos, generalmente, van dotados de unos artificios electromecánicos conocidos por "flopis" (o disquetes), cuya misión es equivalente a la del casete pero con sustanciosas ventajas sobre éste, en cuanto a la rapidez en el manejo de la información contenida en ellos. El Spectrum también los puede manejar.

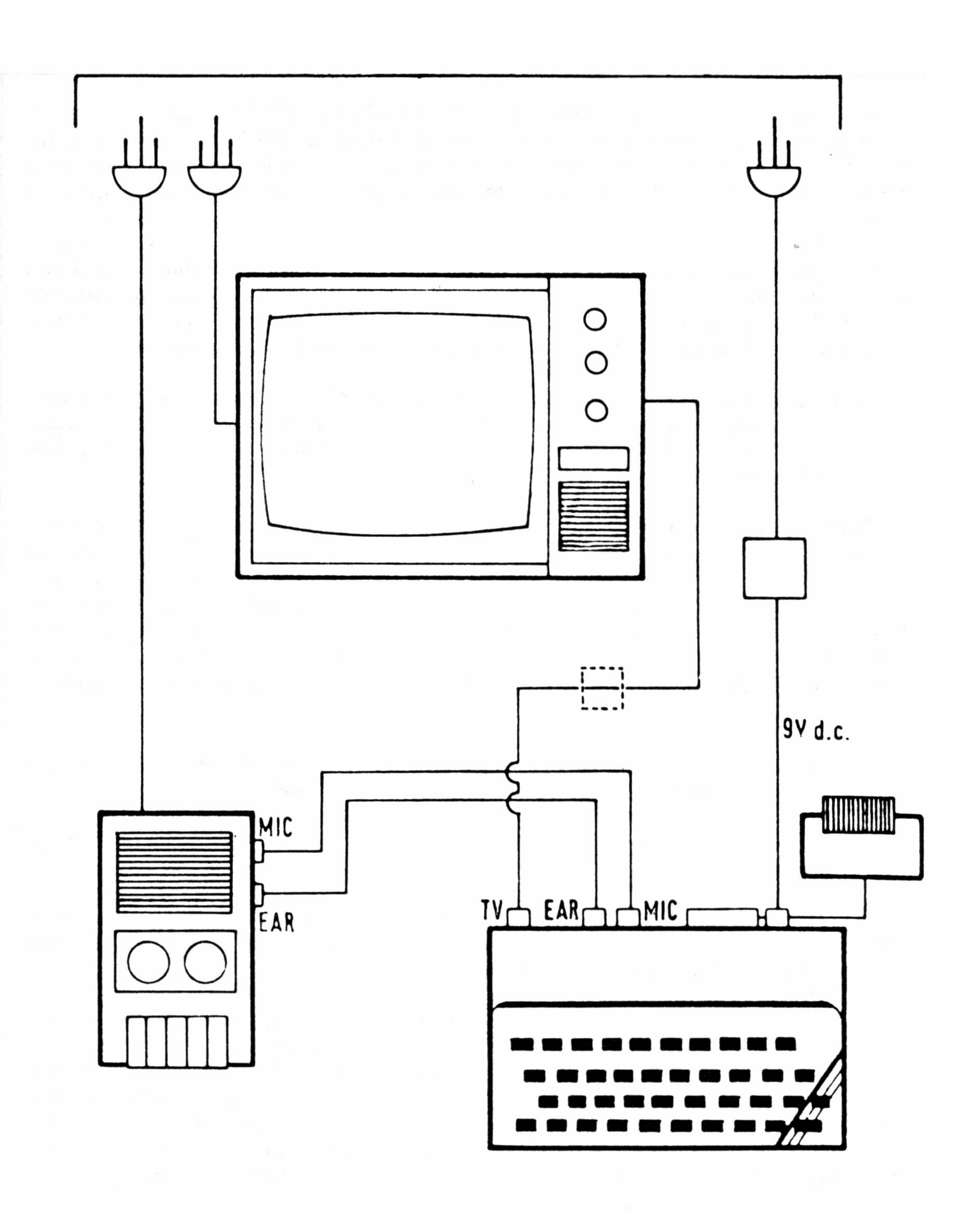


Fig. 1

Esquema de conexiones del Spectrum.

***Impresora.**— Es un dispositivo exclusivamente de salida que permite al usuario proveerse de copias —impresas en papel— de listados de programas, datos o reproducciones del contenido de la pantalla.

La electrónica del Spectrum está constituida esencialmente por un microprocesador modelo Z-80, que cumple la función de UNIDAD CENTRAL DE PROCESO (CPU). Este microprocesador es muy usual en muchos otros microcomputadores. Además están los chips de memoria. Dentro de estos hay que distinguir entre los que "memorizan" la información de forma permanente y salen grabados de fábrica, de acuerdo con las instrucciones del diseñador (memoria ROM), y aquellos otros cuya capacidad de memoria desaparece cuando se corta la corriente eléctrica (memoria RAM).

La memoria ROM del Spectrum es de 16K y contiene el INTERPRETE cuya misión consiste en transformar las instrucciones BASIC —entendibles por los humanos— en microinstrucciones de código máquina —entendibles por el microprocesador Z-80—.

1-3	PROGRAMAS

Hasta ahora hemos oído hablar de ordenadores y programas. De los primeros conocemos su aspecto exterior y, más o menos, la función que cumplen cada uno de los elementos que lo componen —al menos en el Spectrum—, pero con respecto a los segundos —los programas— suele haber confusión o —en muchos casos— desconocimiento total. Quizá sería conveniente aclarar que cualquier computador por muy caro y poderoso que sea es absolutamente estúpido y que sólo gracias a los programas consigue una aparente inteligencia, la cual, además, sólo abarca lo que el programa le permita.

En el caso del Spectrum, y gracias a la memoria ROM, nada más conectar la máquina a la corriente, ya está en condiciones de "interpretar" —de entender— las instrucciones BASIC que reciba, pero, mientras no se le carge en memoria —memoria RAM— el programa adecuado, será incapaz de hacer la O con un canuto.

Supongamos que queremos convertir el ordenador en una máquina capaz de sumar dos números cualesquiera, que el usuario le dé al computador a través del teclado. Si éste intentara que la máquina sumara, sin introducir previamente en su memoria el programa que le diga cómo debe proceder con los números que se le van a dar, nos encontraríamos —todo lo más— con un mensaje de error.

Para conseguir que el ordenador sume hay que enseñarle, y esto sólo se consigue con un programa escrito por alguien que sepa.

Antes de dar el siguiente paso, vamos a ver cómo funciona el cerebro humano en una circunstancia similar.

Si usted le dijera a un muchachito: "Vamos a ver si sabes sumar", de inmediato nuestro interlocutor se preparará para recibir la información adecuada, p. ej. "Cuántas son dos y dos". Dado que el cerebro encargado de realizar la suma esperaba dos números contestará, lógicamente, de inmediato "cuatro".

La forma de preparar el cerebro del ordenador para la suma de dos números es decirle algo así como:

```
10 INPUT A

20 INPUT B

30 LET C = A + B

40 PRINT C
```

Gracias a estas líneas escritas en BASIC el ordenador sabe que debe esperar la entrada de dos números, sumarlos e imprimirlos.

Usted todavía no está en condiciones de interpretar ni de escribir programas en BASIC y, justamente con la intención de enseñarle, está escrita esta SECCION.

Una última consideración. Es evidente la complejidad y capacidad del cerebro humano pero, no obstante, ante situaciones absurdas reacciona con perplejidad. Un ordenador, dada su especialización, se limita a detectar un error. Si en el ejemplo anterior, el muchachito hubiera respondido a su pregunta de "Cuántas son dos y dos?", p. ej., "COLON", con toda seguridad usted habría pasado por unos instantes de confusión hasta llegar a un par de conclusiones posibles. El ordenador, sin embargo, no hubiera dudado en contestar simplemente con un mensaje de error, en el cual le viene a indicar que esperaba un número y ha recibido un caracter.

Dicho esto les voy a presentar a un personaje muy especial.

1-4	YO, EL SPECTRUM

Hola. Soy tu Spectrum.

A partir de ahora, y si sigues mis recomendaciones, aprenderás rápidamente a utilizarme. He aquí la primera: Cuando te pongas delante de mí debes estar relajado y no tener ninguna clase de miedo, ya que, salvo que me maltrates físicamente, es casi imposible que logres estropearme. Por lo demás puedes seguir este curso tan deprisa como quieras, con la condición de que no dudes en volver atrás cuando lo creas necesario.

¿Empezamos?

Yo sé que tú no sabes y, por esa razón, iré poco a poco, pero ahora tendrás que hacer un esfuerzo.

Trata de interpretar el pequeño programa que sigue.

```
1 LET A=127
2 PRINT A
```

Textualmente este programa me viene a decir:

PRIMERO (**1**) DEJA (**LET**) LA VARIABLE **"A"** IGUAL A (=) 127

SEGUNDO (2) IMPRIME (PRINT) EL CONTENIDO DE LA VARIABLE "A"

De este programa debes sacar dos conclusiones. La primera es que debes tener una idea de lo que significan en español las palabras inglesas del lenguaje de programación BASIC. En segundo lugar notarás que mi cerebro es tan ordenado que necesito que te dirijas a mí de acuerdo con unas normas (sintaxis), sin las cuales yo sería incapaz de entender lo que me quieres ordenar.

Estudiemos algunas de estas normas analizando el contenido del programa anterior.

A simple vista podemos ver que está compuesto por dos líneas, pues bien, cada una de ellas se denomina LINEA DE PROGRAMA. La línea 1 ordena LET A=127 y la 2 ordena PRINT A. Estas líneas de programa deben ir de menor a mayor justo en el orden en que tú desees que yo ejecute el programa. Considero muy útil para ti que numeres las líneas de programa de 10 en 10 con lo cual el programa anterior quedaría:

```
10 LET A=127
20 PRINT A
```

De esta forma aún tienes la posibilidad de introducir nuevas líneas de programas, lo cual será de gran utilidad como ya notarás.

Después del número de cada línea de programa existen unas palabras —LET y PRINT— que corresponden a instrucciones del lenguaje de programación BASIC. La instrucción LET asigna un valor a una variable —en nuestro caso le da el valor 127 a la variable A— y la instrucción PRINT me obliga a imprimir en pantalla el valor que previamente se le haya asignado a la variable A.

Las variables pueden ser numéricas o de caracteres según permitan la asignación de números —o fórmulas que finalmente den un número— o por el contrario, sólo acepten textos o cadenas de caracteres. La forma en que yo distingo una de otras es muy sencilla:

Cualquier letra o grupos de letras y números que EMPIECEN POR UNA LETRA me hacen saber que la variable es de tipo numérico. P. ej.:

```
10 LET SPECTRUM16K = 30000
```

Por ser SPECTRUM16K una variable y además ser numérica, el valor que se le puede dar es cualquier número, en este caso le hemos dado el valor 30000. Igualmente podría haber sido 2 ó 1984. Por el contrario, UNA SOLA LETRA SEGUIDA DEL SIMBOLO $ me hace saber que la variable es de tipo alfanumérico —o de caracteres—. P. ej.:

```
10 LET A$="SPECTRUM16K"
```

Por ser A$ una variable y además alfanumérica, el contenido de la misma es cualquier combinación de caracteres SIEMPRE QUE ESTEN ENTRE COMILLAS.

Una vez en este punto vamos a introducir nuestro primer programa en mi electrónico cuerpo. Conéctame a la corriente eléctrica a través de mi transformador... oyes mi zumbidito? Eso significa que estoy "vivo". Ahora ponme en comunicación con un televisor preferiblemente en color y sintoniza mi señal.

Aprieta la tecla ENTER.

Comencemos. En este momento está el cursor en modo K, es decir, espero la entrada de una instrucción BASIC, no obstante te admitiré un número —correspondiente al número de línea— antes de que me des una instrucción, sin que por ello cambie el cursor. Comienza por teclear el primer número de línea —10— y después la tecla L sobre la cual está también la palabra LET, ahora aprieta la letra A. Como observarás la palabra LET te deja automáticamente un espacio entre ella y la letra A. Para acabar la línea sólo queda teclear el símbolo = y el número 127. En este momento tenemos la línea primera del progra-

ma en la pantalla del televisor, pero yo aún no lo he guardado en mi memoria RAM. Para que tal cosa suceda debes apretar, al final de todas y cada una de las líneas de programa, la tecla ENTER. Por tanto, aprieta ENTER.

Fíjate que la línea de programa ha pasado a la parte superior de la pantalla, y un símbolo ">" aparece entre el número de línea de programa y la instrucción en BASIC. A ese símbolo que cumple la función de indicarte en qué instrucción estoy situado, se le conoce como PUNTERO.

Repite el proceso anterior con la segunda y última línea del programa. El programa está ya en mi memoria y listo para ser utilizado, para ello sólo debes saber cuál es la orden correspondiente.

En inglés, la palabra RUN conlleva significados tan dispares como "correr", "poner en movimiento" o "explotar". Así, si tú me das la orden RUN —seguida claro está de la ejecutiva ENTER— yo entenderé que tu deseo es que CORRA el programa o, lo que es igual, que lo ponga en movimiento o en explotación.

Dicho esto, si pruebas a teclear RUN y ENTER, verás aparecer 127 en la esquina superior izquierda de la pantalla.

Más tarde te explicaré el significado de los caracteres que aparecen en la esquina inferior izquierda.

Así los interpreto yo:

Símbolo	**Significado**
=	Me ordena dejar "algo" IGUAL a "algo"
+	Me ordena SUMAR.
—	Me ordena RESTAR.
*	Me ordena MULTIPLICAR.
/	Me ordena DIVIDIR.
↑	Me ordena ELEVAR un número a una potencia. Por ejemplo: 2^3 2 ↑ 3
CUALQUIER NUMERO	Representa ese número.

Ahora fíjate en este ejemplo:

Vamos a hacer un programa para sumar dos números cualquiera, los cuales tú me dirás a través de mi teclado.

Ahora te diré cómo debes "hablarme".

"Espera (INPUT) a que te dé el primer número (A), después, espera (INPUT) a que te dé el otro número (B). A continuación deja (LET) en tu memoria el resultado (C) de sumar los dos números anteriores (A + B). Finalmente, imprime (PRINT) en la pantalla al valor (C) de la suma".

Esto escrito en mi teclado, y de acuerdo con lo estudiado en las páginas anteriores, sería una cosa así:

(No teclees nada en mi teclado hasta que yo te avise).

```
INPUT A
INPUT B
LET C = A + B
PRINT C
```

Si tratas de introducir estas líneas en mi ordenado cerebro no conseguirás nada.

Necesito que me digas claramente el orden en que tengo que actuar.

Esto sería un programa:

(No teclees nada todavía. No seas impaciente.)

Programa	**Comentarios**
1 INPUT A	Primera LINEA DE PROGRAMA.
2 INPUT B	Segunda LINEA DE PROGRAMA.
3 LET C = A + B	Tercera LINEA DE PROGRAMA.
4 PRINT C	Cuarta LINEA DE PROGRAMA

1-5	DE NO SABER NADA... ¡A SABER ALGO!

Estas líneas de programa deben ir de menor a mayor justo en el orden en que tú quieras que yo opere. Cualquier numeración que siga la norma anterior es buena, pero yo te recomiendo que vayas de 10 en 10. Así:

Programa	**Comentarios**
10 INPUT A 20 INPUT B	Numerando las LINEAS DE PROGRAMA de 10 en 10, tienes la posibilidad de introducir nuevas LINEAS con nuevas instrucciones para mí.
30 LET C = A + B	
40 PRINT C	

Tal vez te preguntes qué significan la A, la B o la C. Relee en la página 23 lo referente a variables numéricas.

Dado que A y B pueden ser cualquier número que tú me des por el teclado y, puesto que su suma también puede ser cualquier número, yo necesito asimilarlas a unas VARIABLES NUMERICAS y esas VARIABLES NUMERICAS son las que tú me indiques. En este caso A, B y C.

Ahora enchúfate a la corriente y al televisor. Si dudas, echa un vistazo al comienzo de esta SECCION.

Introduce el programa en mi memoria a través de mi teclado. Sigue el proceso que te indico y analiza los comentarios.

EMPIEZA A TECLEAR:

Proceso	**Comentarios**
	Antes de apretar ninguna tecla aparece una **K** en la esquina inferior izquierda de la pantalla. El **cursor** está representado por esa **K**.
1° escribe 10	Este número lo interpreto como el número de la LINEA DE PROGRAMACION por ser previo a ninguna SENTENCIA.
2° aprieta INPUT	INPUT está encima de la tecla I y es una SENTENCIA U ORDEN. Observa como la **K** se ha transformado en **L**. Esto implica que ya no espero ninguna otra sentencia.

Proceso	Comentarios
3º aprieta A	Por estar el **cursor** representado por una **L**, yo interpretaré que tú quieres justamente la letra A y no ninguno de los otros símbolos que la acompañan en su tecla.
4º aprieta ENTER	Con esta orden yo sé que la línea de programa 10 está completa y que debo guardarla en mi memoria.

En este momento aparece la **K** nuevamente.

5º escribe 20	
6º aprieta INPUT	
7º aprieta B	
8º aprieta ENTER	La línea de programa 20 pasa a mi memoria. El **cursor** se transforma en **K** nuevamente y aparece en el margen inferior izquierdo.
9º escribe 30	
10º aprieta LET	
11º aprieta C	
12º aprieta =	Todas las instrucciones en rojo exigen apretar a la vez SYMBOL SHIFT.
13º aprieta A	
14º aprieta +	
15º aprieta B	
16º aprieta ENTER	La línea de programa 30 pasa a mi memoria.
17º escribe 40	
18º aprieta PRINT	
19º aprieta C	
20º aprieta ENTER	La línea de programa 40 pasa a mi memoria.

1-5	DE NO SABER NADA... ¡A SABER ALGO!

Ya sé lo que quieres que haga y cómo quieres que lo haga.

Vamos a usarlo. Para ello sólo tienes que:

1º Apretar RUN .... CORRELO

2º Apretar ENTER .... ¡YA!

Inmediatamente verás aparecer en la esquina inferior izquierda de tu pantalla una **L**, lo cual te indica que espero el primer número.

Escríbelo y aprieta ENTER. (El 15 por ejemplo.)

Otra vez aparece la **L** a la espera del segundo número.

Escríbelo y aprieta ENTER. (El 5 por ejemplo.)

En el margen superior izquierdo aparecerá el resultado (20 si has introducido los sumandos sugeridos) y en la esquina inferior izquierda tendrás un mensaje mío ϕ OK, 40 : 1, con lo cual te indico que todo salió bien.

En el apéndice 1 puedes ver el código de errores.

Supongamos ahora que ya no quieres sumar. Ahora quieres multiplicar cualquier número por cualquier otro.

Tienes dos opciones: o haces un programa nuevo o modificas éste.

Pero antes haz los **EJERCICIOS DE CALCULO.**

Hasta aquí tu primera lección sobre computación.

Haz los siguientes programas que te permitan respectivamente:

1. Sumar hasta diez números que tú des a través del teclado.

2. Sumar los dos primeros números y restar el tercero.

3. Sumar los dos primeros números y restar el tercero, multiplicando el resultado anterior por un cuarto número.

4. Elevar cualquier número a cualquier potencia.

5. Hallar la raíz cuadrada de un número.

Intenta razonar como "entendería" yo cada cuestión. ¿Una ayudita?

Ayuda al ejercicio 5

"Espera la entrada (INPUT) de un número que llamarás A. Después deja (LET) C igual a la raíz cuadrada (SQR) de C. Después imprime el valor de C".

Ayuda al ejercicio 3

"Espera la entrada (INPUT) de un número que llamarás A. Después espera la entrada (INPUT) de otro número que llamarás B. Después espera la entrada (INPUT) de otro número que llamarás C. Después espera la entrada (INPUT) de otro número que llamarás D.

Después deja (LET) E igual a la suma (+) de A y B.

Después deja (LET) F igual a la resta (-) de E y C.

Después deja (LET) G igual al producto (*) de F por D.

Te aconsejo que antes de empezar a programar escribas el programa en términos parecidos a los ejemplos anteriores.

Una solución a estos ejercicios la tienes en la página siguiente.

Siempre que empieces a teclear un programa no olvides limpiar mi memoria. Para ello tienes dos caminos: o bien desconectas el jack de entrada de corriente, o bien aprietas NEW, ENTER.

1-6	*SOLUCIONES*

Ejercicio 1

Programa	**Comentarios**
1 REM "EJERCICIO 1"	La instrucción REM no afecta al programa. Sólo sirve para recordarte aquello que tú quieras resaltar
10 INPUT N1	Estas instrucciones me ordenan esperar la entrada de 10 números
20 INPUT N2	
30 INPUT N3	
40 INPUT N4	
50 INPUT N5	También podrías haber usado las variables numéricas A, B, C, D, etc. en vez de N1, N2, N3, etc.
60 INPUT N6	
70 INPUT N7	
80 INPUT N8	
90 INPUT N9	
100 INPUT N10	Me ordena "dejar" la variable numérica S igual a la suma de las variables N1, N2, etc.
110 LET S = N1 + N2 + + N3 + N4 + N5 + N6 + + N7 + N8 + N9 + N10	
120 PRINT S	Me ordena "imprimir" el valor de S

RECUERDA:

1. No olvides apretar ENTER después de completar cada línea de programa, p. ej.

   Escribe 10

   Aprieta INPUT

   Escribe N1

   Aprieta ENTER

2. Para correr (RUN) el programa, es decir, para hacerlo funcionar, aprieta RUN, ENTER.

Ejercicio 2

Programa	**Comentarios**
1 REM "EJERCICIO 2"	REM no afecta al programa, sólo sirve para "recordarte" textos.
10 INPUT A	Espero la entrada de la variable numérica A.
20 INPUT B	Espero la entrada de la variable numérica B.
30 INPUT C	Espero la entrada de la variable numérica C.
40 LET D = A + B	Dejo D igual a A + B.
50 LET E = D – C	Dejo E igual a D - C.
60 PRINT E	Imprimo el valor de E.

Ejercicio 3

Agrega las siguientes líneas de programa al programa del ejercicio 2.

35 INPUT F	Espero la entrada de la variable numérica F.
60 LET G = E * F	Al escribir esta línea de programa (y apretar ENTER por supuesto) borras la anterior y "deja" la variable G igual a E * F.
70 PRINT G	Imprimo el valor de G.

Si quieres puedes ir grabando (SAVE) estos programas en un casete y así tendrás una colección de ejercicios grabados y, en todo caso, practicas la grabación (SAVE) y carga (LOAD) de programas.

Ejercicio 4

Programa	Comentarios
1 REM "EJERCICIO 4"	
10 INPUT X	X es la variable que representa la base.
20 INPUT Y	Y es la variable que representa el exponente.
30 LET Z = X ↑ Y	↑ (SYMBOL SHIFT y H) me ordena elevar X a la potencia Y.
40 PRINT Z	

Ejercicio 5

Programa	Comentarios
1 REM "EJERCICIO 5"	
10 INPUT A	
20 LET B = SQR A	SQR me ordena extraer la raíz cuadrada de A. SQR se obtiene así: 1.— SYMBOL SHIFT y CAPS SHIFT con lo cual cambia el cursor a E. 2.— SQR (sobre la tecla.H).
30 PRINT B.	

Ejercicio 3A

Una solución alternativa.

Programa **Comentarios**

```
1 REM "EJERCICIO 3A"
10 INPUT A
20 INPUT B
30 INPUT C
40 INPUT D
50 LET E = (A + B – C) *D
60 PRINT E
```

1-7	BORRANDO Y MODIFICANDO

Tan importante como saber programar es saber borrar y modificar cuando sea necesario.

En inglés DELETE significa "borrar" y EDIT significa "editar". Consiguientemente, cuando queramos borrar usaremos la instrucción DELETE y cuando queramos editar una línea de programa completa para manipularla, bien para borrar parte de ella, bien para introducir algún cambio, utilizaremos la instrucción EDIT. Un ejemplo servirá de explicación.

Un ejemplo servirá de explicación.

Si en ei programa

```
10 LET A = 127

20 PRINT A
```

quisieras cambiar la variable A por la F tendrías que empezar por desplazar el puntero hasta situarlo en la línea 10. En todo caso, el puntero se desplaza arriba y abajo por medio de las teclas 7 y 6, según indican las flechas que están dibujadas justamente encima de las mismas, apretando simultáneamente la CAPS SHIFT en el caso del Spectrum convencional o directamente con las flechas de la última fila de teclas del Spectrum Plus.

Una vez situado el puntero en la línea de programa que desees editar, apretarás la tecla EDIT —de acuerdo con tu tipo de Spectrum— para que yo ejecute la "edición".

Al hacerlo notarás que paso la línea sobre la que está el puntero a la parte inferior de la pantalla, con lo cual te indico que ya estás en condiciones de introducir las modificaciones que consideres necesarias. En el caso propuesto se trata de cambiar la A por la F. Ahora debes desplazar el cursor a la derecha de la A, para lo cual apretarás simultáneamente CAPS SHIFT y 8 ó 5 (⇨, ⇦), y una vez allí proceder al borrado de la letra A, para lo cual apretarás simultáneamente CAPS SHIFT y DELETE o directamente DELETE si tienes un PLUS. Una vez borrada la A debes proceder a insertar la letra F, lo cual sólo requiere apretar la tecla F sin necesidad de mover el cursor. Ya está la línea 10 modificada, efectúa tú las operaciones adecuadas para cambiar la letra A de la línea 20 por la F.

Ejercicio 6. Si tienes el ejercicio 1 grabado en un casete, cárgalo (LOAD) en mi memoria, si no, teclea el programa de nuevo.

Cuando lo tengas listado (LIST, ENTER) en la pantalla, conviértelo de forma que sirva para sumar sólo dos números.

Ejercicio 7. Convierte este último en el programa del ejercicio 3.

Ejercicio 8. Convierte este último en el programa del ejercicio 3A.

Ejercicio 6

Proceso	**Comentarios**
30 ENTER	Cualquier número de línea de programa seguido de ENTER borra el contenido que pudiera tener esa línea de programa.
40 ENTER	
50 ENTER	
60 ENTER	
70 ENTER	
80 ENTER	
90 ENTER	
100 ENTER	

Ahora desplaza el puntero hasta la línea 110 apretando CAPS SHIFT y 6 simultáneamente o directamente la flecha ↓ en el PLUS.

Edita (EDIT) esta línea apretando CAPS SHIFT y 1 o directamente EDIT en el PLUS.

La K está entre 110 y LET. Aprieta CAPS SHIFT y 8 ó → en el PLUS.

La K cambió a L. Sigue desplazando el cursor hasta que la L esté situada a la derecha de N10.

Empieza a borrar (DELETE) hasta que la L se sitúe a la derecha de N2.

Aprieta ENTER.

El programa está modificado. Ya puedes correrlo (RUN, ENTER).

Sólo podrás sumar dos números.

Ejercicio 7

Proceso	**Comentarios**
30 INPUT C	Introduce esta línea (no olvides ENTER)
115 LET E = S - C	Introduce esta línea de programa.

Desplaza el puntero hasta la línea 120.

Edita (EDIT) la línea 120.

Borra la S y sustitúyela por E.

En este punto el programa funciona exactamente igual que el del ejercicio 2, aunque las variables usadas sean distintas.

35 INPUT F	Introduce esta línea de programa.
120 LET G = E * F	Sustituimos la línea 120 (que ya no será útil) por esta otra que cumple la función de multiplicar E por C. (La variable F del ejercicio 3 es aquí la C).
130 PRINT G	Imprime el valor G.

En este ejercicio debes concluir que no importa el número de línea de programación que utilices, con tal de que sea mayor que el número de la línea anterior y menor que el de la línea siguiente: ¡Lo único importante es que yo sepa el orden con que debo realizar las operaciones!

Ejercicio 8

Proceso **Comentarios**

Te aconsejo que cambies todas las variables a las que figuran en el ejercicio 3a.

110 LET E = A + B - C * D	Sustituye la línea 110 por esta otra.

1-8	*SOLUCIONES*

115 ENTER	Borra las instrucciones contenidas en las líneas de programación 1.15 y 120 dejándolas "vacías".
120 ENTER	
130 PRINT E	Es más rápido escribir esta línea (por su longitud) que editarla.

Corre (RUN, ENTER) el programa y haz algunas pruebas.

Para acabar estas prácticas sobre modificaciones y cambios te mostraré un buen truco:

Si quieres "editar" una determinada línea de programación, puedes evitarte la rutina de subir o bajar el "puntero", simplemente apretando LIST seguido del número de la línea de programa que te interese modificar (y ENTER claro). De esta forma el puntero te saldrá en la línea deseada y sólo tendrás que editar.

Prueba LIST 110.

¿Vale?... de nada, de nada.

Un programa claro es un programa fácil de entender. Es muy usual pensar que el programa que hacemos hoy está suficientemente claro para su escritor, de forma que nos da la impresión de que por los siglos de los siglos lo entenderemos. Nada más lejos de la realidad; con toda probabilidad, la próxima vez que lo usemos, si no hemos sido precavidos, tendremos que consumir un tiempo más que razonable para saber cómo utilizarlo o cómo lo hicimos.

En inglés REMARK significa ''observación'' o "nota", de aquí la instrucción REM del lenguaje BASIC, gracias a la cual puedes introducir aclaraciones en un programa sin que el contenido que sigue a las instrucciones afecte para nada a la ejecución del programa.

Volviendo a nuestro programa

```
10 LET A = 127

20 PRINT A
```

y a pesar de su sencillez, sería conveniente introducir un REM de la siguiente forma: 5 REM, este programa es un ejercicio. Una vez tecleada esta línea, el programa quedará como sigue:

```
5 REM este programa es un ejercicio

10 LET A = 127

20 PRINT A
```

Después de hecha esta modificación, todo habrá quedado igual, excepto por el hecho de haber introducido la nota aclaratoria REM de la línea 5, la cual no ha afectado al programa para nada.

En inglés LIST significa LISTA —relación ordenada— y LIST es una instrucción BASIC según la cual me ordenas que te muestre por pantalla el programa que tengo en memoria. En el ejemplo anterior, y después de haber corrido el programa (RUN, ENTER), si quisieras volver a tener en pantalla el programa sólo tendrías que apretar LIST y ENTER.

Las notas REM son fundamentalmente aclaraciones para el programador. Pero no debes olvidar al usuario del programa, que no tiene por que coincidir con su creador y, consiguientemente, necesita saber cómo usar el programa.

Supongamos que vamos a desarrollar un programa que nos sirva para sumar dos números cualesquiera. La estructura fundamental del mismo sería:

```
10 INPUT A

20 INPUT B

30 LET C = A + B
```

La palabra INPUT significa "entrada" y, como instrucción BASIC, me viene a decir "espera la entrada de un dato" con lo cual, cuando yo encuentro una instrucción INPUT, me paro hasta que me introduces por teclado el dato que deseas, seguido de un ENTER para que yo sepa que has acabado de escribir.

El programa anterior lo interpretaría yo, si tuviera tu cerebro, de la siguiente forma:

"Primero (10) espera la entrada de un dato (INPUT A), segundo (20), espera la entrada de otro dato (INPUT B) y tercero (30) deja (LET) la variable C igual a (=) el contenido de la variable A más (+) el contenido de la variable B.".

Si el programa lo dejaras tal cual está, con toda probabilidad no habría quien supiera usarlo, puesto que en la pantalla no aparece ningún mensaje aclaratorio y, además, el resultado de la suma tampoco aparece en la pantalla. En otras palabras, este programa no es útil. Introduciremos algunas líneas aclaratorias:

```
5 REM programa para sumar dos números cuales-
      quiera introducidos por teclado

7 PRINT "Teclea el primer sumando"

10 INPUT A

15 PRINT "Teclea el segundo sumando"

20 INPUT B

30 LET C = A + B

40 PRINT "El resultado de la suma es"

50 PRINT C
```

Teclea este programa en mi teclado y haz algunas pruebas con diferentes pares de números.

CARACTERES

Hasta el momento, prácticamente todos los ejemplos los has hecho utilizando números y variables numéricas, y tal vez pienses que mi habilidad acaba ahí; a lo largo de este capítulo te voy a demostrar lo contrario.

Más arriba te mostré que yo sé que una variable va a ser alfanumérica, siempre que la representes por UNA SOLA LETRA SEGUIDA DEL SIMBOLO $. También te dije que el contenido de una variable alfanumérica debe representarse entre comillas. P. ej.:

```
10 LET S$ = "En un lugar de La Mancha"
```

Las variables alfanuméricas, a diferencia de las numéricas, sólo se pueden sumar. P. ej.:

```
10 LET S$ = "En un lugar de La Mancha"

20 LET B$ = "de cuyo nombre no quiero acor-
            darme..."

30 LET V$ = S$ + B$

40 PRINT V$
```

¡Ojo! No puedes sumar una variable numérica con una alfanumérica, por la misma razón que no podrías sumar JUANITO + 4.

Ejercicio 9. Lista el programa del ejercicio 3 A.

¿Cómo harías para que la pantalla te pregunte?

NUMERO 1 ? (antes de darme el primer número)

NUMERO 2 ? (antes de darme el segundo número)

NUMERO 3 ? (antes de darme el tercer número)

NUMERO 4 ? (antes de darme el cuarto número)

Ejercicio 10. Sobre el resultado del ejercicio 9 ¿Cómo harías para que después de darme cada número yo lo imprima a continuación del NUMERO correspondiente?

Ejercicio 11. Sobre el resultado del ejercicio 9. ¿Cómo harías para que te imprima "EL RESULTADO" y a continuación el valor de E?

¡No mires las soluciones!

Primero inténtalo tú. Estos humanos...

Ejercicio 9

Proceso	**Comentarios**
5 PRINT "NUMERO 1?"	Inserta esta línea
15 PRINT "NUMERO 2?"	Inserta esta línea
25 PRINT "NUMERO 3?"	Inserta esta línea
32 PRINT "NUMERO 4?"	Inserta esta línea

Deja espacio (teclea SPACE) entre la palabra NUMERO y el número que la sigue y entre ? y las ".

Córrelo (RUN, ENTER). Mi respuesta te la doy justo debajo de NUMERO 4?

Ejercicio 10

Proceso **Comentarios**

Lista el programa (LIST, ENTER).

Inserta un punto y coma (;) al final de las líneas de programa 5; 15; 25 y 32.

12 PRINT A	Inserta esta línea
22 PRINT B	Inserta esta línea
37 PRINT D	Inserta esta línea

Corre (RUN, ENTER) el programa modificado con algunos números y compuébalo.

Prueba a quitar los ; de las líneas 25 y 32. ¿Dónde pongo los números con respecto a los dos primeros?

... ¡Ah! Los puntos y comas me atraen...

Ejercicio 11

Proceso	**Comentarios**
Lista el programa (LIST, ENTER)	Si aún está en mi memoria el programa anterior.
120 PRINT "EL RESULTADO"	Inserta esta línea

Corre (RUN, ENTER) el programa con cuatro números cualquiera y prueba esta otra posibilidad:

120 PRINT "EL RESULTADO ="; E

130 ENTER Borro la línea 130

Corre (RUN, ENTER) esta variación con los mismos (u otros) cuatro números.

¡El resultado es igual!

La sentencia PRINT te acepta la posibilidad de concatenar textos y variables numéricas por medio de comas (,) y puntos y comas (;). Así puedes eliminar líneas de programa innecesarias.

Prueba a sustituir el punto y coma de la línea 120 por una coma y corre el programa de nuevo.

Verás que el resultado se ha separado un poco. Tiene una explicación: "La coma me obliga a empezar a imprimir a partir de la columna 16".

Ejercicio 13

Escribe un programa que te pregunte tu nombre y que, al dárselo, imprima:

HOLA tu nombre

RECUERDA:

PROGRAMAR SERA TAN NECESARIO COMO LEER

Ejercicio 12

Haz un programa que te pregunte primero tu nombre, después tu apellido, tu teléfono y por último tu dirección, y que cada respuesta quede registrada en una variable de caracteres.

NOMBRE?

APELLIDO?

TELEFONO?

DIRECCION?

Ejercicio 14

Haz un programa que te pregunte tu profesión y tu edad, y que cada respuesta quede registrada en una variable de caracteres.

Ejercicio 15

Ir directamente a su solución.

Ejercicio 12

Proceso **Comentarios**

```
1 REM "EJERCICIO 12"
10 PRINT "TU NOMBRE"
20 INPUT N$
30 PRINT "HOLA"; N$
40 PRINT "RECUERDA:"
50 PRINT "PROGRAMAR SERA TAN NECESARIO COMO LEER".
```

Ejercicio 13

Programa **Comentarios**

```
1 REM "EJERCICIO 13"
10 PRINT "NOMBRE?"
20 INPUT N$
30 PRINT "APELLIDO?"
40 INPUT A$
50 PRINT "TÈLEFONO?"
60 INPUT T$
```

Aunque la entrada sea un número yo lo voy a considerar como una cadena de caracteres, ya que lo asocio a una variable de caracteres denominada T$

```
70 PRINT "DIRECCION ? "
80 INPUT D$
```

Sigue los pasos que te indico a continuación y aprenderás algo realmente interesante.

Proceso	**Comentarios**
1° RUN, ENTER	Con esta instrucción me ordenas que corra el programa, es decir (como ya sabes) que lo haga funcionar. PERO ADEMAS PONGO A CERO (DEJO VACIAS) TODAS LAS VARIABLES NUMERICAS Y DE CARACTERES.
2°	Contesta a las preguntas: nombre, apellido, etc...
3° PRINT A$, ENTER	Con esta instrucción me ordenas que imprima el valor de A$

Te daré tu nombre en la esquina superior izquierda.

Prueba con las otras variables. (PRINT T$, p. ej.)

AHORA NO BORRES ESTE PROGRAMA. No aprietes RUN. INTRODUCE A CONTINUACION EL PROGRAMA DEL

Ejercicio 14

Proceso	**Comentarios**
1 REM "EJERCICIO 14"	
10 PRINT "PROFESION?"	
20 INPUT Z$	Observa que utilizo una variable de caracteres (Z$) distinta a las usadas en el ejercicio 12.
30 PRINT "EDAD?"	
40 INPUT Y$	Vale el comentario de la línea 20
50 PRINT "NOMBRE:";N$	Yo mantengo en mi memoria el valor de N$ al margen de que haya en mi memoria un programa nuevo... siempre que no me hayas desconectado o apretado RUN o NEW, o uses las mismas variables en el programa nuevo.
60 PRINT "APELLIDOS:";A$	Yo mantengo el valor de las variables, mientras tú no las cambies.

```
 70 PRINT "TELEFONO:"; T$
 80 PRINT "DIRECCION:"; D$
 90 PRINT "PROFESION:"; Z$
100 PRINT "EDAD:"; Y$
```

ATENCION: No corras el programa con RUN.

A partir de ahora usa GOTO (seguido del primer número de línea de programa), y ENTER como siempre.

RECUERDA: Cuando usas RUN, ENTER yo correré el programa, pero ADEMAS TE BORRO EL CONTENIDO DE LAS VARIABLES.

Por tanto, aprieta en este caso, GOTO 1 (o cualquier número anterior a 10, ya que 10 es la primera línea de programa útil), a continuación ENTER.

Contesta a las preguntas del programa.

Cuando lo hayas hecho, e inmediatamente, imprimiré toda la información que contienen las variables de caracteres que has utilizado y que tengo en memoria.

Me puedes listar como siempre (LIST, ENTER).

Tendrás oportunidad de apreciar las ventajas de todo esto.

De momento te aconsejo que guardes (SAVE) este programa en un casete, con los valores que tengan las variables. Lo llamaremos "EJERCICIO 14".

La utilizaremos en el ejercicio 15.

Ejercicio 15

Supongo que has guardado el programa anterior.

Habrás notado lo práctico que es el cuenta vueltas de tu magnetofón para saber dónde empieza cada programa...

Bien, carga (LOAD) el ejercicio 14.

¿Ya lo has hecho?

RECUERDA: Si aprietas RUN, ENTER te quedas con el programa en mi memoria pero "borro" el contenido de las variables.

Vamos a hacer pruebas:

Prueba	Resultado
1 Aprieta GOTO 50	Impresión en pantalla de tus datos.
2 Aprieta GOTO 1	Te pregunto tu profesión. Si me das otra distinta, sustituiré el valor de la variable Z$.

Igual sucederá con Y$.

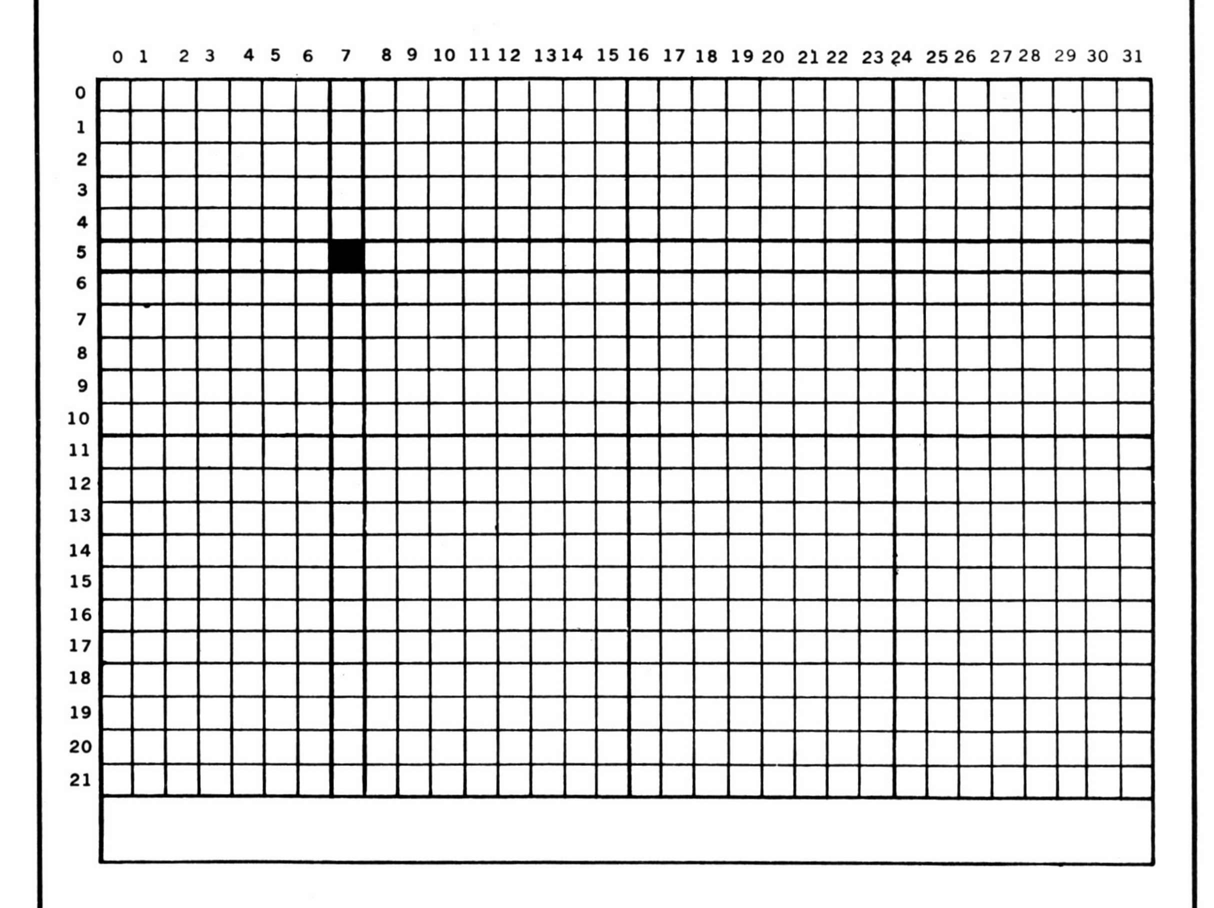
Varía estos datos.

Observa el resultado.

No grabes en el casete esta modificación.

1-12	LA PANTALLA

Imagínate que la pantalla de tu televisor es una pizarra, en la cual yo voy a escribir las respuestas a las preguntas que tú me hagas. Bien, pues como yo no tengo ojos como tú, necesito tener puntos de referencia a los cuales dirigirme para poder imprimir donde tú me indiques; con este fin, yo considero la pantalla dividida en 24 líneas y 32 columnas, numeradas a partir del 0 y de la esquina superior izquierda.



Como ya sabes, PRINT significa IMPRIMIR y, como fácilmente se puede deducir, PRINT AT significa IMPRIME EN. En base a esto, si yo encuentro a lo largo de un programa la instrucción:

PRINT AT 5, 7; "A"

Sé que debo imprimir en la LINEA 5 y COLUMNA 7 la letra A.

Debes recordar que las dos últimas líneas —la 22 y la 23— me las reservo yo para comunicarme contigo y no están disponibles para ti.

CLS es una contracción de CLEAR SCREEN que significa DESPEJAR LA PANTALLA. Si tú me das la instrucción CLEAR yo te dejaré la pantalla en blanco, pero no temas por el programa y los datos que tenga en memoria, yo no los olvidaré.

PRINT TAB es fácil de entender si estás familiarizado con el uso del tabulador pero, en todo caso, quiere decirme que imprima en la misma línea en que estoy trabajando, pero en la columna que me indica el número que sigue al TAB. Veamos un ejemplo:

Teclea directamente, para que veas el efecto del TAB de forma inmediata, la siguiente instrucción: PRINT TAB 5; "A". Podrás apreciar que he impreso la letra A en la columna 5 de la línea que esté disponible y le corresponda. Haz algunas pruebas seguidas con PRINT TAB y comprenderás el significado de este último párrafo.

PAUSE, evidentemente, significa PAUSA. PAUSE es una instrucción que me ordena mantener en pantalla todo lo que en ella está impreso, durante el tiempo que tú me digas, después seguiré ejecutando el programa. PAUSE 50 equivale a una pausa de un segundo y PAUSE 500 a 10 segundos. Prueba el efecto de este programa:

```
 10 PRINT "En este mundo traidor"
 20 PAUSE 100
 30 CLS
 40 PRINT "NADA ES VERDAD NI MENTIRA"
 50 PAUSE 100
 60 CLS
 70 PRINT "sólo el computador"
 80 PAUSE 50
 90 CLS
100 PRINT "TRABAJA Y TU NO LE OBLIGAS"
```

1-13	LA PANTALLA

Bueno, después de este "spectral" verso es muy práctico saber cómo usar las comas (,) y los puntos y coma(;). Compara como se produce la impresión en los dos programas que siguen y que aparentemente son iguales.

10 PRINT "RO" .... 10 PRINT "RO";

20 PRINT "MARBELLA" .... 20 PRINT "MARBELLA";

30 PRINT "VE" .... 30 PRINT "VE";

Cuando corras ambos programas (RUN y ENTER) verás que el primero te imprime en la pantalla:

RO

MARBELLA

VE

y el segundo:

ROMARBELLAVE

Luego, puedes deducir que los puntos y comas me atraen hasta tal punto que, siempre que una orden de impresión acabe en ; yo me veré obligado a imprimir la siguiente orden justo a continuación de aquélla.

Si, por el contrario, detrás de una orden de impresión me encontrara una coma (,) afectaría a las siguientes de esta forma"

10 PRINT "NOMBRE",

30 PRINT "APELLIDOS",

30 PRINT "*"

Una misma instrucción PRINT puede servir para imprimir diferentes textos. El ejemplo anterior se podría escribir también de esta manera, sin que la impresión variara:

10 PRINT "NOMBRE", "APELLIDOS", "*"

Si utilizáramos este criterio en el ejemplo donde se empleaban los puntos y comas, también sería válido. Comprueba que no hay variaciones:

10 PRINT "RO"; "MARBELLA"; "VE"

Habrás observado que después de terminar la ejecución de cada programa, te envío un mensaje del tipo:

0 OK, 90:1

Los caracteres anteriores a la coma (,) te indican el tipo de error, si lo hay —ϕ OK significa que he corrido el programa y no ha habido ningún error—. Los caracteres posteriores a la coma definen la línea en la cual me encuentro y, finalmente, la sentencia dentro de esa línea.

Ejercicio 16

Haz un programa que te imprima automáticamente un * en la esquina superior izquierda de la pantalla, otro en el centro y otro en la esquina inferior derecha. (Usando PRINT AT).

(No olvides que las dos últimas líneas de la pantalla no están disponibles para tí.)

Repite el programa poniendo a continuación del * las vocales AEIOU.

Ejercicio 17

Sabrías hacer el programa anterior usando PRINT y PRINT TAB.

Ejercicio 18

Carga (LOAD) el "EJERCICIO 13" en mi memoria.

Cambia el programa (SIN ALTERAR EL CONTENIDO DE LAS VARIABLES) de forma que imprima los textos a la izquierda de la pantalla y el contenido de las variables de caracteres a partir de la columna 20 y en la misma línea (Usa PRINT TAB).

Ejercicio 19

Cambia la impresión de las variables del programa anterior para que imprima a partir de la columna 16 (Usa la coma (,)).

Ejercicio 16

Programa	Comentarios
1 REM "EJERCICIO 16"	
10 PRINT "*"	También vale PRINT TAB 0; "*" o PRINT AT 0,0; "*". Con una columna de variación.
20 PRINT AT 10, 16; "*"	Tambien podrías haber escrito: 20 PRINT, "*"
30 PRINT AT 21, 32; "*"	

Ejercicio 17

Programa	Comentarios
1 REM "EJERCICIO 17"	
10 PRINT "*"	
20 PRINT	Una instrucción PRINT sin nada a continuación me obliga a dejar una línea en blanco
30 PRINT	
40 PRINT	
50 PRINT	
60 PRINT	
70 PRINT	Este procedimiento de espaciar las líneas de impresión en pantalla es incómodo y consume una gran capacidad de memoria.
80 PRINT	
90 PRINT	
100 PRINT	
110 PRINT TAB 16; "*"	

```
120 PRINT
130 PRINT
140 PRINT
150 PRINT
160 PRINT
170 PRINT
180 PRINT
190 PRINT
200 PRINT
210 PRINT
220 PRINT
230 PRINT TAB 31; "*"
```

Acostúmbrate a usar PRINT TAB, PRINT AT y siempre que puedas punto y coma (;) y coma (,).

Ejercicio 18

Programa	**Comentarios**
50 PRINT "NOMBRE";	Cambia la línea 50 por ésta.
TAB 20; N$	Fíjate como se ha colocado aquí TAB.
60 PRINT "APELLIDOS:";	
TAB 20; A$	
70	
80	
90	Cambia tú estas líneas
100	

Cuando hayas acabado las modificaciones y quieras comprobar el resultado, deberás apretar GOTO 50; de esta forma, yo te correré el programa desde la línea 50 sin borrar las variables y sin hacerte las preguntas que van en las líneas de programa anteriores.

¿Has seguido estas instrucciones?

Bueno, pues ahora aprieta GOTO 1 y... ¿A ver cómo te las ingenias para que te dé la impresión por pantalla... sin alterar el contenido de las variables?

1-15 SALTOS

Normalmente, mi proceso de trabajo consiste en ejecutar las instrucciones una detrás de otra, justamente como ya sabes, pero hay ocasiones en que te interesa que se altere este sistema, saltando incondicionalmente a una línea de programa distinta a la siguiente. Para que yo sepa que debo hacer ésto, debes decirme "VETE A la línea de programa número tal". Existe una instrucción BASIC que cumple esta función: GOTO, que significa exactamente VETE A.

Veamos cómo funciona el siguiente programa:

```
10 REM salto incondicionado

20 PRINT "PRIMER SUMANDO?"

25 INPUT A

30 PRINT "SEGUNDO SUMANDO?"

50 INPUT B

60 LET C = A + B

70 PRINT "LA SUMA ES"; C

80 GOTO 20
```

Como verás, la línea 80 es la única nueva y es de clara interpretación en tanto en cuanto me obliga a ir a la línea 20, donde volveré a iniciar el ciclo y, mientras tú no hagas algo, yo seguiré ininterrumpidamente llegando a la línea 80 y volviendo a la 20. Esto es un BUCLE.

Cuando yo esté metido en un bucle tendrás dificultades para conseguir que yo obedezca una instrucción LIST. Primero te daría un error de interpretación, si la variable que espero es numérica, con lo cual saldría del bucle y ya estaría en condiciones de aceptar cualquier orden. Si por el contrario estoy a la espera de una variable alfanumérica, tendrías que utilizar la instrucción BREAK (SPACE y ENTER) antes de pulsar LIST.

Este epígrafe necesita pocas explicaciones y lo mejor es hacer los EJERCICIOS CON SALTOS INCONDICIONALES.

Ejercicio 19

Haz un programa que imprima las vocales (A, E, I, O, U) una debajo de otra en la pantalla, y, hasta que ésta se llene.

Usa instrucciones PRINT y GOTO.

Ejercicio 20

¿Se puede cambiar el valor numérico de la instrucción GOTO por una variable numérica?

Ejercicio 21

Haz el ejercicio 19 usando PRINT AT, GOTO y, ,

Ejercicio de interpretación

¿Cómo funcionaría este programa?

```
10 LET A = 1
20 PRINT A
30 GOTO 10000
1000 GOTO 1000
```

Ejercicio 19

Programa	Comentarios
10 PRINT "A"	A
20 PRINT "E"	E
30 PRINT "I"	I
40 PRINT "O"	O
50 PRINT "U"	U
60 GOTO 10	A
	E
	I
	O
	U
	A
	E
	I
	O
	U
	A
	E
	I
	O
	U
	A
	E

Ejercicio 20

Sí se puede. Cambia la línea 60 del programa anterior por: 60 GOTO X y ahora introduce LET X = 10 (sin número de línea), y aprieta ENTER.

El programa correrá igual que antes, ya que a X le hemos dado el valor 10. Prueba haciendo:

LET X = 20

LET X = 30

LET X = 40

LET X = 50

Ejercicio 21

Programa	**Comentarios**
5 LET C = 0	5. Se define el valor de la variable C.
10 PRINT AT C, 0;	10. La doble coma obliga a un salto de línea, imprimiendo por tanto para cada valor de C:
"A", , "E", , "I", , "O", , "U"	A E I O U
20 LET C = C + 5	20. Deja a C igual al valor anterior que tuviera más cinco y, así obtenemos:
	A 1
	E 2
	I 3

1-16	*SOLUCIONES*

Programa **Comentarios**

O 4

U 5

A 6

E 7

I 8

O 9

U 10

" "

" "

```
30 GOTO 10
```

Yo puedo tomar decisiones siempre y cuando me des las instrucciones oportunas con arreglo a la sintaxis que exige el BASIC.

Entre humanos os podéis decir cosas como éstas: "SI te llamas Pepe ENTONCES vete a Caracas". Obviamente, yo no entendería una orden así... pero sí una muy aproximada. Supongamos que me quieres enviar a la línea 540 si la variable A es igual a 10 en el transcurso del siguiente programa:

```
10 REM tomando decisiones

20 PRINT "Primer factor?"

30 INPUT X

40 PRINT "Segundo factor?"

50 INPUT Y

60 LET A = X * Y

70 IF A = 10 THEN GOTO 540

80 PRINT "EL PRODUCTO ES"; A

90 GOTO 20

540 PRINT "ESTOY EN LA LINEA 540 PORQUE LA VARIABLE A
   VALE "; A: GOTO 90
```

Un par de cosas llaman la atención en este programa. La primera es la línea 70, según la cual me indicas que en el caso —y sólo en el caso— de que la variable A sea igual a 10 me dirija a la línea 540: "SI A = 10 ENTONCES VETE A 540", esto sería un salto condicionado, ya que implica un cambio en el normal proceso de lectura de instrucciones si se cumple una condición previa. El segundo detalle interesante está en los dos puntos (:) que aparecen antes del GOTO de la línea 540.

Yo, el Spectrum, permito que en una sola línea de programa entre más de una instrucción, para lo cual debes respetar esta regla: "Después de cada instrucción completa se ha de colocar el símbolo ":", de esta forma, yo sabré dónde acaba una y empieza otra instrucción. Siguiendo esta indicación puedes confeccionar un programa de esta manera:

```
10 REM Líneas multisentencia
20 PRINT "Escriba su nombre por favor": INPUT N$: PRINT
"GRACIAS"; N$
```

Volviendo a nuestra sentencia IF ... THEN es conveniente saber que cualquier operador aritmético es susceptible de ser usado. Sobre las teclas Q, W, E, R, T y L te ofrezco mis posibilidades. Veamos un ejemplo:

```
10 REM EJERCICIO CON < =
20 PRINT "DAME UN NUMERO"
30 INPUT A
40 IF A < = 10 THEN PRINT A: GOTO 20
50 PRINT A/2
```

Esto me viene a decir que si el número que me das es menor o igual a 10 lo debo imprimir, en cualquier otro caso también lo imprimiría, pero dividido por 2.

Antes de pasar a los ejercicios debo volver a recordarte mi habilidad para trabajar con caracteres y variables alfanuméricas. Estoy perfectamente capacitado incluso para compararlas, debido a que, para mí, las letras tienen un "peso" y consiguientemente puedo saber si una está antes que otra.

Estudia este ejemplo:

```
10 REM Comparando letras
20 PRINT "Dime una letra"
30 INPUT A$
40 PRINT "Dime otra letra"
50 INPUT B$
60 IF A$ < B$ THEN GOTO 90
70 PRINT B$; "ES MAYOR QUE "; A$
80 GOTO 20
90 PRINT A$; "ES MENOR QUE"; B$
```

Compara las letras del abecedario entre sí, luego palabras con palabras y, finalmente, números.

Ejercicio 22

Desarrolla un programa que te diga:

PASE o ESPERE, según el color azul o rojo de un semáforo. El color lo introduces tú por el teclado.

Ejercicio 23

Para ser candidata a Miss Universo se debe medir más de 1,70 metros y pesar menos de 60 Kilos.

¿Podrías hacer un programa en el que dando el nombre, estatura y peso de cada aspirante te diga si reúne o no los requisitos mínimos?

Ejercicio 24

¿Qué tendrías que hacer,para que la Srta. Juanita del Enchufe y Alto Voltaje fuera admitida, sea cual fuere su talla y peso, en el concurso de belleza del programa anterior?

Ejercicio 25

¿Qué instrucciones pondrías al principio de un programa para que sólo funcionara con las personas que conozcan la "clave secreta"? En el ejercicio anterior, por ejemplo.

Ejercicio 26

Escribe un programa que ponga en orden alfabético cuatro palabras cualquiera.

Ejercicio 22

Programa **Comentarios**

```
10 REM                    SEMAFORO
20 PRINT "color del semáforo? (azul/rojo)".
30 INPUT A$
40 IF A$ <> "AZUL" THEN GOTO 70
50 PRINT PAPER 1; INK 1; AT 11, 13; "PASE"
60 GOTO 80
70 PRINT PAPER 2; INK 1; AT 11, 13; "ESPERE"
80 PRINT AT 20,0; "OTRO ENSAYO? (s/n)"
90 INPUT R$
100 IF R$ = "SI" THEN CLS : GOTO 20
110 CLS : PRINT AT 0,0; "ADIOS"
120 STOP
```

Ejercicio 23

Programa **Comentarios**

```
 5 PRINT "NOMBRE?"
```

Observa como la operación lógica OR implica que la instrucción THEM se cumple cuando alguna de las condiciones IF es cierta.

```
10 INPUT N$
11 PRINT N$
15 PRINT "TALLA?",
20 INPUT T
21 PRINT T
25 PRINT "PESO ?",
```

```
30 INPUT P
31 PRINT P
35 IF T < 1,7 OR P > 60 THEN GOTO 70
40 PRINT N$, "ADMITIDA"
45 PRINT AT 21,0; "OTRA CANDIDATA ? "
50 INPUT O$
55 IF O$ = "NO" THEN STOP
60 CLS
65 GOTO 5
70 PRINT N$, "RECHAZADA"
75 GOTO 45
```

Ejercicio 24

Modifica la línea 35:

```
35 IF T < 1,7 OR P > 60, THEM IF N$ <> "JUANITA DEL ENCHUFE"
THEN GOTO 70
```

Ejercicio 25

```
1 PRINT "CLAVE ? "
2 INPUT C$
3 IF C$<> "CUALQUIER COMBINACION DE CARACTERES" THEN STOP
5 PRINT "NOMBRE ?"
10 INPUT N$
11 PRINT N$
```

1-19

REPETICION

La instrucción FOR-TO/NEXT es aparentemente complicada pero de suma utilidad para determinadas aplicaciones, y tal vez mejor que ninguna explicación previa sea estudiar un ejemplo sencillo y comentarlo posteriormente.

```
10 REM ejemplo FOR-TO/NEXT
20 REM TABLA DE MULTIPLICAR POR DOS
30 FOR Y = 0 TO 10
40 LET A = 2 * Y
50 PRINT A
60 NEXT Y
70 STOP
```

Haciendo caso omiso de las REM de las líneas 10 y 20, este programa tiene el siguiente significado para mí: "PARA el valor 0 de la variable Y, y HASTA que esta variable alcance el valor 10, DEJA la variable A igual a 2 por el valor actual de la variable Y; a continuación IMPRIME el valor de A y, finalmente, da a la variable Y el PROXIMO valor y vuelve a la instrucción FOR de la línea 30, donde se repetiría el ciclo anteriormente expuesto pero con Y = 1. Y así sucesivamente hasta que la variable Y alcanzara el valor 10, con cuyo valor se consumiría el último ciclo y pasaría a la instrucción 70."

En la práctica mi proceso de trabajo consistiría en darle a la variable Y el valor 0, multiplicarlo por 2, imprimir el resultado, al llegar a la instrucción NEXT hacer Y = 1 y volver a la línea 30 donde empezaría otra vez multiplicando por dos este nuevo valor de Y, imprimiéndolo y, al llegar a NEXT daría el siguiente valor a Y (Y = 2) comenzando otra vez todo el proceso. Y así una y otra vez hasta que el siguiente valor fuera 11, con lo cual la variable Y tomaría un valor superior al impuesto por la instrucción TO, que en nuestro caso es 10, con lo cual yo seguiría a la instrucción 70.

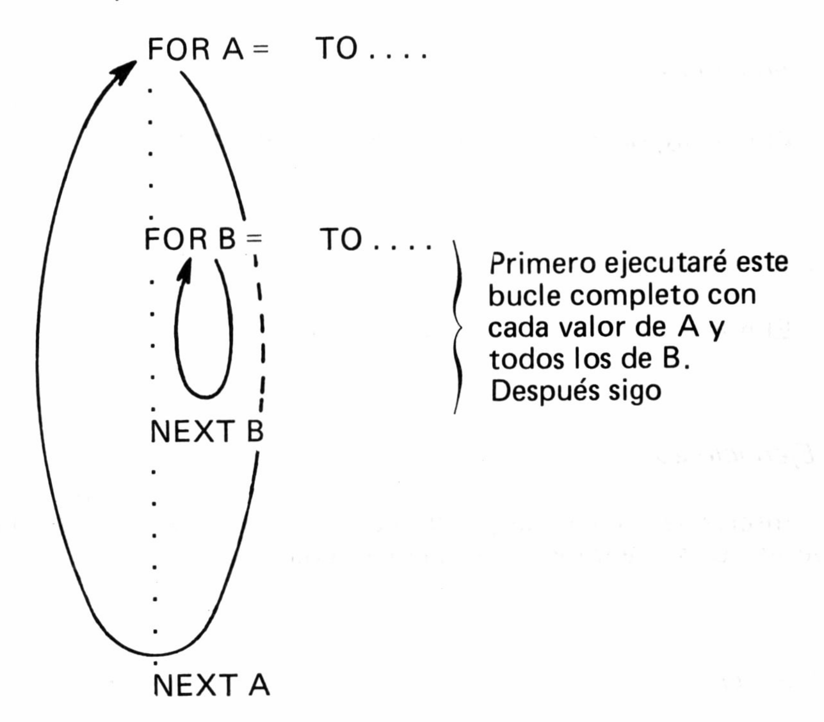
En ocasiones interesa que la cadencia con que la variable va tomando valores sea distinta a la unidad; para ello tienes disponible la instrucción STEP. Si en el programa anterior cambiásemos la línea número 30 por la siguiente:

```
30 FOR Y = 0 TO 10 STEP 2
```

el proceso de trabajo sería exactamente igual excepto por el hecho de que al llegar a la instrucción NEXT los valores que iría tomando la variable Y variarían de dos en dos, ya que así lo impone el STEP 2 que hemos introducido.

En este tipo de sentencias, la variable (Y en este caso) **sólo puede ser una letra.**

Dentro de un bucle FOR-TO/NEXT puedes meter otro bucle FOR-TO/NEXT con la única condición de que no se crucen. Observa el siguiente esquema:



¿Recordamos la tabla de multiplicar por 3, por 4 y por 5?

En este tipo de bucles (anidados) funciono de la siguiente forma:

Cojo el primer valor del primer bucle y con este valor me recorro completamente el bucle anidado; una vez realizado esto, cojo el siguiente valor del primer bucle y vuelvo a recorrerme el bucle anidado, y así sucesivamente.

Ejercicio 26

Usando el ciclo FOR/NEXT haz un programa que te imprima la columna 0 con la palabra "columna".

Ejercicio 27

El mismo, pero que se imprima en la columna 12.

Ejercicio 28

El mismo, pero que llene toda la pantalla.

Ejercicio 29

Inserta las líneas de programa suficientes como para conocer el número de veces que se ha impreso la palabra "columna".

Ejercicio 30

Escribe un programa que te dé en pantalla los números impares del 1 al 50.

Ejercicio 31

Modifica el programa anterior de forma que sólo responda la suma de los números impares del 1 al 50.

Ejercicio 32

Escribe un programa con un "parpadeo" lento.

Ejercicio 33

Con dos bucles FOR/NEXT enlazados crea una imagen de puntos en pantalla similar a ésta:

Cómo interpretas este programa:

```
5 FOR E = 1 TO 2
10 LET E = 1
20 LET A = 2
30 LET C = E + A
40 PRINT C
50 NEXT E
```

Ejercicio 26

Programa | **Comentarios**

```
5 FOR X = 0 TO 21
10 PRINT "COLUMNA"
20 NEXT X
```

Se imprime la palabra "columna" 22 veces a partir de la columna ϕ y línea tras línea.

Ejercicio 27

Programa | **Comentarios**

```
5 FOR X = 0 TO 21
10 PRINT TAB 12; "COLUMNA"
20 NEXT X
```

TAB 12 obliga a comenzar en la columna 12

Ejercicio 28

Programa | **Comentarios**

```
5 FOR X = 0 TO 21
10 PRINT "COLUMNA";
20 NEXT X
30 GOTO 5
```

El ; obliga a imprimir una palabra tras otra. La línea 30 obliga a repetir el ciclo FOR/NEXT hasta que aparezca el mensaje **scroll?**. Si aprietas cualquier tecla excepto N o BREAK, el proceso se repetirá.

Ejercicio 29

Programa | **Comentarios**

```
4 LET C = 0
```

Hemos incorporado un contador

```
5 FOR X = 0 TO 21
10 PRINT "COLUMNA"
11 LET C = C + 1
20 NEXT X
30 GOTO 5
```

Ejercicio 30

Programa **Comentarios**

```
10 FOR X = 1 TO 50 STEP 2
20 PRINT X,
30 NEXT X
```

Ejercicio 31

Programa **Comentarios**

```
5 LET C = 0
10 FOR X = 1 TO 50 STEP 2
11 LET C = C + X
30 NEXT X
35 PRINT AT 21,0;
"VALOR DE LA SUMA..."; C
```

Ejercicio 32

11 FOR X = 0 TO 10 — Los bucles FOR/NEXT no se pueden cruzar, pero, evidentemente, sí se pueden suceder.

20 PRINT AT 13, 15; "BASIC" — Escribe la palabra BASIC en negativo.

50 NEXT X

60 FOR X = 1 TO 10

70 PRINT AT 13, 15; "BASIC"

80 NEXT X

90 GOTO 10

Ejercicio 33

Programa	**Comentarios**
10 FOR F = 0 TO 21	Observa cómo varía el valor de entrada del segundo bucle (C) con el valor de entrada del primero (F) y consiguientemente, el lugar de arranque de PRINT
20 FOR C = F TO 31	
30 PRINT AT F, C; "."	
40 NEXT C	
50 NEXT F	

Parece claro en razón de lo dicho hasta aquí y por los ejercicios realizados que un programa es una rutina. Constantemente repito el mismo proceso siempre que me pidas que ejecute el mismo programa. Hay muchas ocasiones que, dentro de un mismo programa, se repiten algunas líneas de programa una y otra vez. Para evitar estas repeticiones innecesarias, el BASIC nos provee de unas instrucciones (GOSUB y RETURN), gracias a las cuales sólo es necesario escribir esas líneas específicas una sola vez.

Si asociamos programa a rutina, será inmediato concluir que una subrutina es un programa que está dentro de una rutina general o programa.

En inglés, en el supuesto que quisieras mandar a alguien a una subrutina, dirías "go subrutine" (ve a subrutina) y para volver de ella dirías "return" (vuelve). Y de aquí las instrucciones BASIC que yo interpreto como órdenes para salir del programa principal, seguir mi proceso de trabajo en la subrutina (GOSUB) y, cuando encuentre la instrucción RETURN, volver al programa general justo en línea que siga a la GOSUB.

Estas dos instrucciones —complementarias— son muy fáciles de aplicar y un ejemplo te ayudará a entenderlas.

```
10 REM Aplicando subrutinas
20 PRINT "Primer sumando?"
30 INPUT A
40 GOSUB 200
50 PRINT "Segundo sumando?"
60 INPUT B
70 GOSUB 200
80 LET C = A + B
90 PRINT "La suma es "; C
100 GOSUB 200
110 STOP
200 FOR X = 0 TO 31
```

```
210 PRINT "*";
220 NEXT X
230 RETURN
```

Este programa mantiene la estructura fundamental del que venimos usando, pero, aparte de los GOSUB que has introducido y que comentaremos a continuación, observarás una sentencia STOP, cuya misión fundamental es DETENER la ejecución del programa al llegar a la línea donde el programa general acaba propiamente.

Cuando un programa es detenido por un STOP puede ponerse en marcha nuevamente con una instrucción CONTINUE —en el teclado CONT sobre la tecla C—.

Como ya sabes, yo leeré las instrucciones 10, 20... pero al llegar a la 30 me encuentro con una orden clara de dirigirme a la línea 200. Una vez allí interceptaré y ejecutaré las líneas 200, 210, 220... y al llegar a la 230 una tajante instrucción RETURN me obliga a volver al programa principal y justamente a la línea 50 que es la siguiente a la línea que contenía el GOSUB que me obligó a ir a la subrutina. Teclea el programa y córrelo.

Ejercicio 34

¿Se te ocurre alguna forma de combinar el programa del ejercicio 33, con el ejercicio 32, para que, convirtiendo éste en una subrutina de aquél, la palabra BASIC cruce parpadeando la pantalla de izquierda a derecha?

Ejercicio 35

Escribe un programa para obtener la tabla de multiplicar por 5, separando cada número por una línea de asteriscos.

```
5 POR 0 =  0
***********
5 POR 1 =  5
***********
5 POR 2 = 10
***********
5 POR 3 = 15
***********
5 POR 4 = 20
***********
5 POR 5 = 25
***********
5 POR 6 = 30
***********
5 POR 7 = 35
***********
5 POR 8 = 40
***********
5 POR 9 = 45
***********
5 POR 10=50
***********
```

Ejercicio 36

Haz de nuevo el ejercicio 1 con una subrutina, de forma que te pregunte por pantalla: "NUMERO?" antes de la entrada de cada número y te dé las "GRACIAS" después. Limítalo a cinco sumandos.

Ejercicio 37

Cómo aprovecharías las subrutinas del ejercicio anterior para usarlas en las del ejercicio 3A.

NOTA: Pon las subrutinas sólo si te ahorran trabajo personal e instrucciones.

Ejercicio 34

Programa | **Comentarios**

```
1 FOR F = 0 TO 21
```

En este ejercicio se anidan dos bucles FOR/ NEXT con una subrutina.

```
20 FOR C = F TO 31

30 GOSUB 90

40 NEXT C

50 NEXT F

90 FOR Y = 0 TO 2

100 PRINT AT F, C; "BASIC"
```

Escribe BASIC en negativo

```
110 CLS

140 PRINT AT F, C; "BASIC"

160 NEXT Y

170 RETURN
```

Ejercicio 35

```
10 FOR X = 0 TO 10

20 LET A = X * 5

30 PRINT "5 POR "; X; "="; A

40 GOSUB 100

50 NEXT X

100 PRINT "**********"

110 RETURN
```

Ejercicio 36

Programa | **Comentarios**

```
 5 GOSUB 100
```

Las subrutinas conviene ponerlas después de la última instrucción del programa.

```
10 INPUT N1
15 GOSUB 200
16 GOSUB 100
20 INPUT N2
25 GOSUB 200
26 GOSUB 100
30 INPUT N3
35 GOSUB 200
36 GOSUB 100
40 INPUT N4
45 GOSUB 200
46 GOSUB 100
50 INPUT N5
55 GOSUB 200
60 LET S = N1 + N2 + N3 + N4 + N5
70 PRINT " LA SUMA ES "; S;" DE NADA"
80 STOP
100 PRINT "NUMERO?"
110 RETURN
```

```
200 PRINT "GRACIAS"

210 RETURN
```

Ejercicio 37

En general puedes aprovechar cualquier programa como una subrutina de cualquier otro. Basta con que no olvides la instrucción RETURN y no duplicar las variables con lo cual perderías información o provocarías errores.

1-23	ARRAYS? COMO SE UTILIZAN

Una guerra... no importa contra quien.

La batalla va a comenzar.

Los jefes y oficiales de las fuerzas a tu mando se reúnen contigo, que es quien tiene la responsabilidad de la operación, para recibir instrucciones respecto a la forma de distribuir las tropas.

Todos esperan, ansiosos, tus órdenes.

Los miras y les dices:

"Caballeros, que cada cual se ponga donde le plazca".

El combate comienza necesitándose urgentemente la colaboración de la artillería, tus ayudantes buscan a los artilleros y no los encuentran.

Igual sucede con los infantes, los blindados, la intendencia...

Salvo que tu eventual enemigo quede desconcertado por tu "estrategia", el resultado de esta imaginaria batalla está claro.

Lo usual es preparar un orden de combate que permita hacer maniobrar a las distintas unidades según convenga y de forma rápida y precisa.

Vamos a hacer un esquema.

DISPOSICION ESQUEMATICA DE LAS TROPAS

			FUERZAS DE COMBATE	JEFE AL MANDO
		SECTOR 1	INFANTERIA 1	CORONEL VALIENTE
ENEMIGO	ZONA DE COMBATE	SECTOR 2	INFANTERIA 2	CORONEL BRAVO
		SECTOR 3	CARROS 1	CAPITAN ACERO
		SECTOR 4	ARTILLERIA 101	CAPITAN OBUS
IMAGINARIO		SECTOR 5	INFANTERIA 3	CORONEL AUDAZ
		SECTOR 6	INFANTERIA 4	CORONEL VIVO

En este supuesto hay 6 sectores de combate.

Si hubiera 3000 y te resultara imprescindible saber qué fuerzas hay en el sector 1121 y quién las manda, quizás te puedas imaginar el lío.

Para hacer el ejemplo fácil de manejar por mí y de seguir por tí lo dejáremos como está en el esquema.

RECUERDA:

Una variable de caracteres queda representada por UNA SOLA LETRA seguida del símbolo $.

1-24	ESQUEMA DE LAS TROPAS

En función de lo que ya conoces, el significado de F$ representa una variable de caracteres y nada impide que la hagas igual a cualquier cadena de caracteres Por ejemplo:

F$ = "INFANTERIA 1"

Por idénticas razones podrías escribir:

I$ = "CORONEL VALIENTE"

y de esta forma podrías seguir:

```
A$  = "INFANTERIA-2"
B$  = "CORONEL BRAVO"
C$  = "CARROS-1"
E$  = "CAPITAN ACERO"
F$  = "ARTILLERIA 101"
G$  = "CAPITAN OBUS"
H$  = "INFANTERIA-3"
I$  = "CORONEL AUDAZ"
K$  = "INFANTERIA-4"
L$  = "CORONEL VIVO"
```

Y así, continuarías hasta que...

Se acabaran las letras y te quedaras sin variables y poco o nada habrías conseguido.

Vamos a usar un ARRAY de caracteres.

Anticipo que hay dos tipos de ARRAY:

— numéricos (trabajan sólo con variables numéricas)

— caracteres (trabajan con variables de caracteres)

Compara las instrucciones anteriores con las que siguen. Estudia las diferencias y los comentarios.

Comentarios

F$ (1) = "INFANTERIA-1"... A la variable de caracteres F$ la hemos convertido en F$ (1) esto, en argot, sería decir: EFE DOLAR DE UNO O EFE DOLAR SUBUNO, es decir, una variable de caracteres con subíndice (representado por (1)), es igual, en este caso a INFANTERIA-1.

Observa que la cadena de caracteres INFANTERIA-1 tiene 12 caracteres.

F$ (2) = "INFANTERIA-2"... EFE DOLAR DE DOS es igual a una cadena de caracteres de 12 caracteres.

F$ (3) = "CARROS-1"... También podrías decir que EFE DOLAR DE TRES tiene 8 caracteres.

F$ (4) = "ARTILLERIA-101".... También: EFE DOLAR SUB 4 tiene 14 caracteres.

F$ (5) = "INFANTERIA-3"... F$ (5) tiene 12 caracteres.

F$ (6) = "INFANTERIA-4"... F$ (6) tiene 12 caracteres.

En este caso F$ es el prefijo y el número entre paréntesis es el sufijo.

Compara las instrucciones anteriores con las que siguen.

Comentarios

F$ (1,12)... EFE DOLAR DE UNO, DOCE O EFE DE DOLAR SUBUNO, DOCE.

Esto quiere decir que la variable de caracteres F$ (1) **puede** tener **hasta** 12 caracteres.

F$ (2,12). La variable de caracteres F$ (2) puede tener hasta 12 caracteres.

F$ (3,8)... La variable F$ (3) puede tener hasta 8 caracteres.

F$ (4, 14)... La variable F$ (4) puede tener hasta 14 caracteres.

F$ (5, 12)... F$ (5) puede tener hasta 12 caracteres.

F$ (6,12)... F$ (6) puede tener hasta 12 caracteres.

De todas las variables de caracteres F$ anteriores la que admite más caracteres es, en este caso, la F$ (4,14).

El número total de variables cuyo prefijo sea F$ es 6.

Si yo me encontrara en una línea de programa, la instrucción DIM F$ (6,14) inmediatamente reservaría en mi memoria espacio suficiente para aceptar 6 VARIABLES DE CARACTERES QUE EMPIECEN POR F$ Y QUE PUEDAN TENER HASTA 14 CARACTERES.

En nuestra "batalla" hemos asimilado DIM F$ (6,14) a 6 "sectores de combate" en los cuales hay unas tropas que puedes denominar como quieras siempre que tengan hasta 14 caracteres. En este caso F$ es el prefijo.

Si escribieras DIM J$ (6,15), y siempre referido a nuestro esquema principal, estarías en condiciones de darme 6 "jefes al mando" cuya denominación no tenga más de 15 caracteres. En este caso J$ es el prefijo.

Bueno, pues esto son ARRAYS de caracteres:

"Conjunto de variables de caracteres con el mismo prefijo y el mismo límite de cantidad de caracteres que puede aceptar".

Ejemplo:

Si yo me encuentro la instrucción:

DIM L$ (100,25)

yo la interpretaría como:

"Dimensiona un conjunto de 100 variables de caracteres que todas tengan el prefijo L$ y que sean capaces de aceptar hasta 25 caracteres cada una".

Bueno. ¿Y qué? Dirás tú.

No seas impaciente y sigamos con nuestro ejemplo.

A través de las sentencias DIM yo sé las dimensiones y las denominaciones de los ARRAYS que voy a utilizar... pero estos ARRAYS están vacíos porque no contienen información. Están disponibles.

En nuestra "batalla" tú utilizas F$ para introducir en mi memoria, por sectores, las Fuerzas de Combate y J$ para introducir los Jefes de las mismas.

¿Cómo me lo debes decir?

Empecemos a programar usando ARRAY de caracteres:

Programa	**Comentarios**
30 DIM F$ (6.14)...	Me ordena dimensionar un ARRAY de caracteres de 6 variables y hasta 14 caracteres cada una, y cuyo prefijo es F$. De momento está vacío. No contiene nada.
40 DIM J$ (6.15)...	Me ordena dimensionar un ARRAY de caracteres de 6 variables y hasta 15 caracteres cada una, y cuyo prefijo es J$. De momento está vacío. No contiene nada.
50 PRINT "SECTOR?";	Me ordena imprimir "SECTOR?". (deja 2 espacios entre ? y ").
60 INPUT S...	Espero la entrada de un número, que en nuestro caso deberá estar comprendido entre 1 y 6, ambos inclusive, y que corresponderá a "un sector de combate".
70 PRINT S...	Imprimo el valor de S después de "SECTOR"? puesto que el ; de la línea 50 me obliga a ello.
80 PRINT "FUERZAS?";	Imprimo "FUERZAS?" dejando dos espacios entre ? y ". El ; obliga a la siguiente impresión a ponerse justo a continuación.
90 INPUT F$ (S)...	Espero la entrada de una variable de caracteres. Por tener el prefijo F$, yo introduciré este valor en el ARRAY, dimensionado en la línea 30. La S entre paréntesis tendrá el valor que haya recibido en la línea 60. El número de caracteres está fijo y definido en la línea 30, no se puede pasar de 14. Esto quiere decir que si el valor de S (que tú darás al correr el programa, a través del teclado) fuera por

	ejemplo 2, yo estaré esperando que me fijes el valor de la variable F$ (2.14). El ARRAY se va llenando. Va conteniendo información.
100 PRINT F$ (S)	Imprimo el texto que hayas introducido en la línea 90 y justo a continuación de lo impreso por la línea 80.
110 PRINT "JEFE?";	Imprimo JEFE? dejando dos espacios entre ? y ". El ; obliga a la siguiente impresión a ponerse justo a continuación.
120 INPUT J$ (S)	Vale el mismo comentario de la línea 90, pero referido al prefijo J$ y al ARRAY dimensionado en la línea 40.
130 PRINT J$ (S)	Imprimo el texto que hayas introducido en la línea 120 y justo a continuación de lo impreso por la línea 110.
140 GOTO 50	Me ordena: VETE a la línea 50.

NOTA: A partir de ahora, cuando quieras correr un programa no aprietes RUN, ENTER; es mejor que sigas este procedimiento:

1. GOTO (número de la línea de programa desde donde quieres hacer funcionar el programa).
2. ENTER
   Ya te explicaré la razón más adelante.

¿Has introducido ya el programa?

Teclea GOTO 30, ENTER.

En la pantalla aparece: SECTOR?

Aprieta 1 y ENTER.

Ahora te pregunto: FUERZAS?

Teclea: INFANTERIA–1 y ENTER.

JEFE?

Teclea: CORONEL VALIENTE.

(Observa como, si dejas un espacio entre CORONEL y VALIENTE yo te imprimo CORONEL VALIENT).

¿Sabes por qué? Porque sólo admito 15 caracteres.

Vuelvo a preguntarte SECTOR?, después FUERZAS?, JEFE?

Deberás seguir este ciclo 6 veces que son los sectores de combate, y que, por otro lado, es el número máximo de variables de caracteres que voy a admitir (DIM F$ (6,14) y DIM J$ (6,15)).

Prueba a meter un sector 7 y te daré un mensaje de error, con lo cual te digo que el subíndice 7 está fuera de la dimensión que me has dado y que por esa razón estoy parado en la línea 90.

Lista el programa de nuevo (LIST, ENTER). ¿Lo has hecho?

Probablamente pienses que ya no se qué fuerzas hay en el sector 3 y quién las manda.

Vamos, vamos... teclea.

Proceso	**Comentarios**
PRINT F$ (3)	Me ordena imprimir el valor que tengo en memoria para la variable F$ (3). y ahora
PRINT J$ (3)	Lo mismo para la J$ (3).

A veces pienso que me menosprecias.

Si acaso se te ocurriera correr el programa apretando RUN, ENTER, te obedecería claro, pero borraría el contenido de todas las variables; por eso, es mejor usar GOTO seguido del número de la primera línea de programa útil.

Este programa lo puedes llamar "BATALLA".

Guarda (SAVE) este programa, tal como está, en un casete.

Desconéctame.

Ahora vuelve a cargarme (LOAD). ¿Ya?

Aprieta:

PRINT F$ (5) (ENTER, claro).

Y después

PRINT J$ (5) (¡No te olvides nunca de ENTER!).

¡Toda la información está guardada en el casete! Tanto el programa como el contenido de las variables.

Lista el programa (LIST, ENTER).

Para tener una visión general de los sectores, las fuerzas y sus jefes, deberás insertar estas líneas:

Programa	**Comentarios**
150 FOR S = 1 TO 6 160 PRINT S; " "; F$ (S); " "; J$ (S)	Con este bucle FOR NEXT. La variable S coge todos los valores del 1 al 6, imprimiendo S, dejando un espacio, imprimiendo F$ (S), dejando un espacio e imprimiendo J$ (S).
170 NEXT S	

Para obtener esta información te verás obligado a teclear:

GOTO 150, ENTER, ya que si teclearas

GOTO 50, ENTER, el programa correría hasta la línea 140 donde la sentencia GOTO 50 me haría empezar de nuevo.

Prueba ambas alternativas.

Graba en el casete este programa sobre el anterior. Lo puedes llamar "BATALLA".

Manejar el ARRAY con soltura te será de mucha utilidad.

Repasa bien este capítulo y haz los EJERCICIOS CON ARRAYS DE CARACTERES.

Ejercicio 38

Dimensiona un ARRAY de caracteres que permita alojar las variables de caracteres de hasta 5 caracteres cada una.

Ejercicio 39

Dimensiona otro ARRAY para 5 variables de hasta cuatro caracteres.

Ejercicio 40

¿Qué significan estas líneas de programa?

```
10 DIM A$ (15,5)

15 DIM B$ (15,5)
```

Ejercicio 41

Haz un programa que te permita dar valores automáticamente a las 4 variables del ARRAY DIM A$ (4,7).

Ejercicio 42

Interpreta este programa:

```
10 DIM A$ (4,7)

20 DIM B$ (4,7)

30 FORX = 1 TO 4

40 LET B$ (5–X) = A$ (X)

50 NEXT X
```

Ejercicio 43

Escribe un programa que ponga en orden alfabético no más de 4 palabras de hasta 7 letras cada una.

Ejercicio 38

DIM A$ (4,5)

Ejercicio 39

DIM A$ (5,4)

Ejercicio 40

Los ARRAYS en general se dimensionan al principio de los programas.

En este caso tenemos dos ARRAYS de caracteres (símbolo $) denominados A y B pudiendo contener cada uno de ellos 15 variables de hasta 5 caracteres.

Ejercicio 41

Programa	**Comentarios**
10 DIM A$ (4,7)	Si alguna de las 4 variables tuviera más de 7 caracteres, yo ignoraría el exceso.
20 FOR X = 1 TO 4	
30 INPUT A$ (X)	
40 NEXT X	

Ejercicio 42

Se transfieren los valores de las variables contenidas en el ARRAY A$ al B$ de tal forma que A$ (1) = B$ (4), A$ (2) = B$ (3), y así sucesivamente.

Ejercicio 43

Programa

```
10 DIM A$ (4,7)
20 FOR X = 1 TO 4
30 INPUT A$ (X)
40 NEXT X
40 NEXT X
50 IF A$ (1) < = A$ (2)
THEN GOTO 90
60 LET B$ = A$ (1)
70 LET A$ (1) = A$ (2)
80 LET A$ (2) = B$
90 IF A$ (2)< = A$ (3)
THEN GOTO 140
100 LET B$ = A$ (2)
110 LET A$ (2) = A$ (3)
120 LET A$ (3) = B$
130 GOTO 50
140 IF A$ (3) < = A$ (4)
THEN GOTO 190
150 LET B$ = A$ (3)
160 LET A$ (3) = A$ (4)
```

Comentarios

Con las cuatro primeras instrucciones introducimos las palabras que valoran las variables de ARRAY.

Con el resto de las instrucciones se comparan todas las variables del ARRAY, ordenándolas, según el "peso" relativo que yo tengo de ellas y que corresponde a su orden alfabético.

```
170 LET A$ (4) = B$
180 GOTO 90
190 PRINT A$ (1)
200 PRINT A$ (2)
210 PRINT A$ (3)
220 PRINT A$ (4)
```

Supongamos ahora que acabas de comprar una tienda de ropa y me quieres usar como caja registradora.

Supongamos también que, de momento, tienes tres artículos en stock: artículo 1, artículo 2 y artículo 3; y los precios unitarios de venta son 10, 20 y 30 europesetas.

Para que yo te pueda ser útil debes desarrollar un programa que me haga funcionar como una caja registradora.

Con los conocimientos de BASIC que tienes hasta ahora, éste podría ser un primer programa:

Programa	**Comentarios**
1 REM "CAJA REGISTRADORA"	Con estas instrucciones doy valor (0) a las variables A, B y C, las cuales representan el importe de la venta por artículo (1, 2, 3).
3 LET A = 0	
4 LET B = 0	
5 LET C = 0	
10 CLS	Limpio la pantalla
15 PRINT "ARTICULO N?"	Estas dos instrucciones piden el número de identificación del artículo (1, 2 ó 3)
20 INPUT N	
25 PRINT "UNIDADES VENDIDAS ?"	Estas dos instrucciones piden la cantidad vendida del artículo anterior.
30 INPUT V	
35 LET P1 = 10	Se definen los precios de los artículos.
40 LET P2 = 20	
45 LET P3 = 30	

50 IF N = 1 THEN LET A = P1 * V	Según el artículo vendido determino el importe de la venta
55 IF N = 2 THEN LET B = P2 * V	
60 IF N = 3 THEN LET C = P3 * V	
65 LET T = A + B + C	T representa el total sucesivo.
70 PRINT "OTRO ARTICULO? S/N"	Esta bifurcación me lleva al principio del programa si hay otro artículo vendido o nos da el total por venta si hemos llegado al final.

```
75 INPUT R$

80 IF R$ = "S" THEM GOTO 10

85 PRINT AT 21, 0; "TOTAL VENTA", T
```

Es evidente que, si para una tienda con tres artículos has necesitado un programa de 17 líneas de programa, para el caso de 3.000 artículos hubiera sido una tarea de "chinos" confeccionar el programa.

Para evitarte esta tediosa tarea, el BASIC dispone de los ARRAYS numéricos.

Si recuerdas el caso de los ARRAYS de caracteres, los subíndices representaban: el primero, el número de variables de caracteres que lo integraban; y el segundo, el número de caracteres máximos que forman cada variable.

En los ARRAYS numéricos, por el contrario, los subíndices sólo fijan una, y sólo una, de las variables numéricas que integran el ARRAY. Nada más.

El ARRAY numérico se representa en la misma forma que los caracteres, pero sin el símbolo $.

Todo ARRAY numérico debe ir precedido por la sentencia DIM —letra— seguida por un paréntesis donde se indique el número de variable que contenga. Por ejemplo:

ARRAY	Variables que comprende
P (6)	P (1), P (2), P (3), P (4), P (5), P (6)
P (2,3)	P (1,1), P (1,2), P (1,3) P (2,1), P (2,2), P (2,3)
P (3,2)	P (1,1), P (1,2) P (2,1), P (2,2) P (3,1), P (3,2)

En los números anteriores los tres ARRAY pueden contener el mismo número de variables:

P (6) . . . . 1 * 6 = 6 variables

P (2,3) . . . . 2 * 3 = 6 variables

P (3,2) . . . . 3 * 2 = 6 variables

El programa de la caja registradora podría quedar transformado, usando un ARRAY numérico y siguiendo las instrucciones anteriores, en:

Programa	Comentarios
1 LET C = 0 5 DIM P (3)	Dimensiono un ARRAY numérico de tres variables numéricas denominadas P. (P (1), P (2) y P (3)).
	Aquí almacenaré los precios de los tres artículos (1, 2, 3)
10 DIM T (3)	Dimensiono otro ARRAY de las mismas características que el anterior pero denominado T.
15 FOR X = 1 TO 3 20 PRINT "PRECIO DEL ARTICULO"; X	Este bucle FOR/NEXT cogerá, como ya sabes, los valores 1, 2 y 3 sucesivamente. Preguntando por los precios de los artículos 1, 2 y 3. De esta manera P (X) cogerá los valores P (1) = 10, P (2) = 20 y P (3) = 30

25 INPUT P (X)	
30 NEXT X	
35 CLS	Hecho lo anterior, limpio la pantalla. Las variables numéricas están definidas.
40 PRINT "ARTICULO NUMERO?"	Aquí propiamente empieza el programa de caja registradora, y, pregunto por el número del artículo vendido.
50 INPUT N	
55 PRINT "UDS. VENDIDAS?"	Determina el número de unidades vendidas del artículo anterior.
60 INPUT V	
65 IF P (N) <> 0 THEN LET T = P (N) * V	Según el artículo vendido (N) determino el precio de venta.
70 LET C = C + 1	Contador cuyo valor será primero 1, luego 2 y finalmente 3.
75 LET T (C) = T	Instrucción que me sirve para poder dar valor a las tres variables que configuran el ARRAY definido en la línea 10.
80 PRINT "OTRO ARTICULO?" 85 INPUT R$	Esta bifurcación me lleva al principio del programa si hay otro artículo vendido o me indica que sume el valor de las tres variables del ARRAY DIM T e imprima el resultado.

```
90 IF R$ = "S" THEN GOTO 35

95 PRINT AT 21,0; "TOTAL VENTA",

T (1) + T (2) + T (3)
```

A simple vista puede que te parezca más largo este programa que el anterior; no obstante, con ligeras modificaciones, podrías tenerlo dispuesto para cualquier cantidad de artículos.

Debes recordar que, una vez valoradas las variables del ARRAY de precios P (1), P (2) y P (3), no hace falta que cada vez que uses el programa les vuelvas a dar valores; para ello no aprietes RUN y ENTER, ya que así borras el contenido de todas las variables.

Basta con que teclees GOTO 40 y ENTER y así pasa a correr el programa desde la pregunta "ARTICULO NUMERO?".

Si te interesa conocer el contenido de alguna variable, P (2) por ejemplo, bastaría con que apretaras: PRINT P (2), ENTER y yo te diré el valor de esa variable.

Ejercicio 44

Dimensiona un ARRAY numérico que pueda contener hasta 10 variables numéricas.

Ejercicio 45

Escribe un programa que te permita valorar las 10 variables numéricas del ARRAY anterior.

Ejercicio 46

Dimensiona un ARRAY numérico bidimensional que pueda contener hasta 10 variables numéricas.

Ejercicio 47

Escribe un programa que te permita valorar las 10 variables numéricas del ARRAY anterior. Guarda en un casete el programa y el ARRAY valorado.

Ejercicio 48

Inserta las instrucciones oportunas al programa anterior para que te pregunte si quieres conocer el valor de alguna de las variables del ARRAY, y, en caso afirmativo, te pregunte la fila y la columna, imprimiendo el contenido de esa variable en la pantalla.

Ejercicio 49

Carga el programa del ejercicio 47 y, sin modificar el valor de las variables, adapta el listado de forma que, automáticamente, se valoren las variables del ARRAY DIM W (2,5) con la condición de que los sucesivos valores del ARRAY A se multipliquen por 2 si son mayores que 50 y se eleven al cuadrado si son menores y se transfieran, con esta condición, al ARRAY W.

Ejercicio 44

```
DIM A (10)
```

Ejercicio 45

```
10 DIM A (10)
20 FOR X = 1 TO 10
30 INPUT A (X)
40 NEXT X
```

Ejercicio 46

```
DIM A (2,5)
```

Ejercicio 47

```
10 DIM A (2,5)
20 FOR X = 1 TO 2
30 FOR Y = 1 TO 5
40 PRINT "VALOR"; X ; " , " ; Y ; "DEL ARRAY NUMERICO A"
50 INPUT A (X,Y)
60 NEXT Y
70 NEXT X
```

Ejercicio 48

```
10 DIM A (2,5)
20 FOR X = 1 TO 2
30 FOR Y = 1 TO 5
40 PRINT "VALOR"; X; ", ", Y, "DEL ARRAY NUMERICO A"
50 INPUT A (X, Y)
60 NEXT Y
70 NEXT X
100 PRINT AT 20,0; "QUIERES CONOCER EL VALOR
DE ALGUNA VARIABLE?"
110 INPUT R$
115 CLS
120 IF R$ = "N" THEN STOP
130 PRINT "FILA?"
140 INPUT F
145 PRINT F
150 PRINT "COLUMNA?"
160 INPUT C
165 PRINT C
170 PRINT A (F.C)
180 GOTO 100
```

Ejercicio 49

```
5 DIM W (2,5)
10 DIM A (2,5)
20 FOR X = 1 TO 2
30 FOR Y = 1 TO 5
52 IF A (X,Y) < = 50 THEN GOTO 57
53 LET W (X, Y) = A (X,Y) * 2
57 LET W (X,Y) = A (X, Y) ↑ 2
60 NEXT Y
70 NEXT X
```

ALGUNOS CONSEJOS POSTERIORES

El código de errores te orientará casi siempre para saber por qué no quiero seguir funcionando.

A veces el mensaje de error te lo daré yo colocando una ? parpadeando en el lugar donde se haya producido el fallo de sintaxis o forma de escribir en el lenguaje BASIC. En estos casos bastará con que desplaces el cursor hasta donde está situada la ? y rectifiques oportunamente. P. ej.:

10 PRINT J"	al apretar ENTER aparecerá la ? a la izquierda de la J, indicando que faltan comillas.

No obstante, habrá ocasiones en que, aun sabiendo que el error es de sintaxis, no sabrás cuál es la causa. En esta situación lo mejor, y probablemente más rápido, será borrar la línea y buscar otro modo de darme la instrucción.

El consejo anterior es igualmente válido para el caso de otro tipo de errores.

No olvides que yo entiendo sólo las sentencias cuando me las das a través de la tecla correspondiente. No trates de escribirlas. P. ej.:

10 PRINT	Apretar solo P...	BIEN
10 PRINT	Apretar P	
	R	
	I	MAL
	N	
	T	

Y ... hasta aquí el curso de iniciación para principiantes. ¿Seguimos?

Sección II

Fichero de palabras del BASIC de SINCLAIR

VOCABULARIO Y SU SINTAXIS

Al llegar a esta sección del libro usted conoce los fundamentos del BASIC y, lo que es más importante, su cerebro se ha estructurado de forma que comprenda el funcionamiento de un ordenador al emplear un lenguaje de alto nivel. A partir de este momento avanzar a lo largo de esta segunda parte le resultará muy sencillo, ya que el objeto de la misma es aumentar su "vocabulario".

Algunas de las instrucciones que vamos a estudiar son exclusivas del Spectrum y difícilmente las encontrará, con idéntica aplicación, en otros ordenadores pero, una vez conocidas éstas, le será fácil interpretar otras cuya función sea similar.

El BASIC, como *lenguaje de programación,* está compuesto por un **vocabulario** y una **sintaxis** determinados.

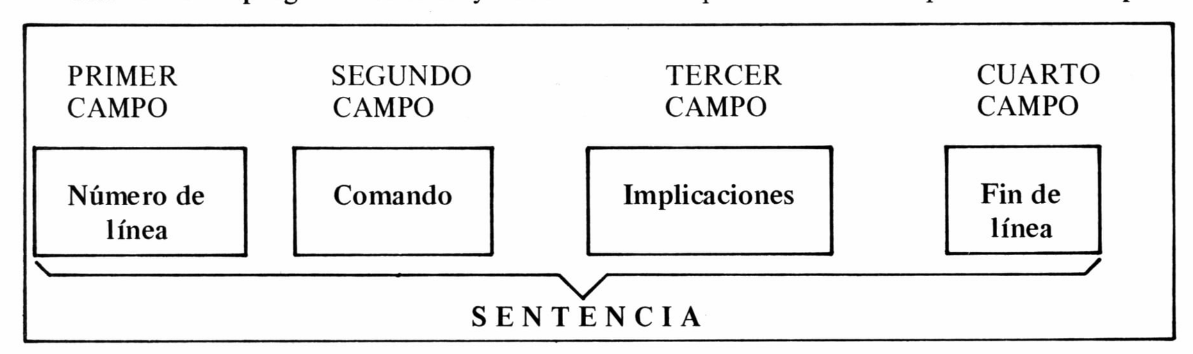
El *vocabulario* es el conjunto de *palabras* que representan a los diferentes **comandos** que acepta e interpreta.

La *sintaxis* es el conjunto de normas que se deben seguir para escribir correctamente un programa.

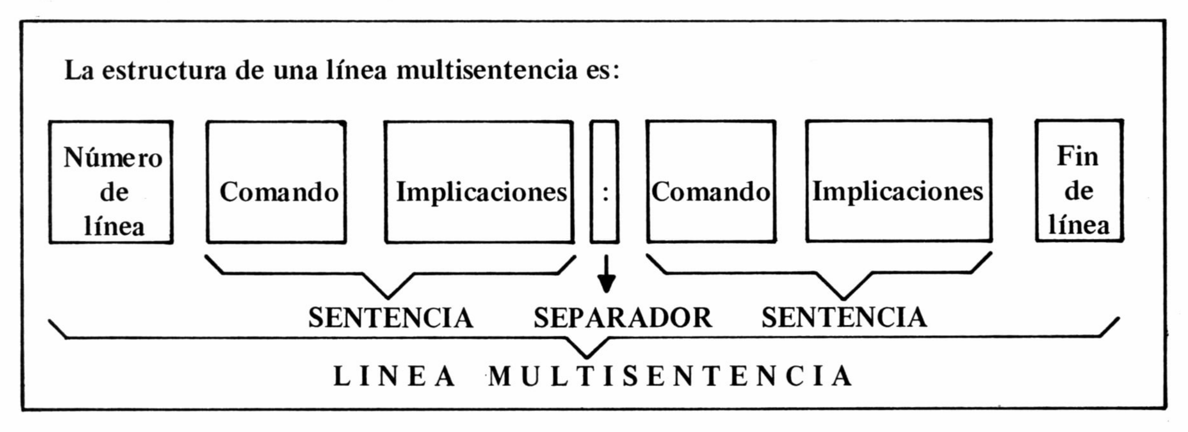
Esta sección está dedicada a estudiar, una a una y con ejemplos, todas las *palabras* que componen el BASIC de Sinclair y sus reglas de aplicación a excepción de las dedicadas a los gráficos, el manejar byte que verán en la SECCION III.

Las primeras páginas introducen al lector en la estructura de las fichas y en alguno de los conceptos que en ellas se utilizan.

Una **línea de programa** constituye una sentencia que está formada por **cuatro campos:**



También existen las líneas de programa que tienen varias sentencias: son **líneas multisentencia,** y responden al esquema anterior, pero separando una sentencia de otra mediante **separadores** y que son *los dos puntos* (:).



Vamos a estudiar la línea de programa explicando cada uno de sus *campos.*

Gracias al **primero,** se establece la secuencia de lectura de las líneas de programa, siendo, pues, imprescindible que los números de línea vayan de menor a mayor, justo en el orden que el programador desee que se ejecuten.

Esto no implica que los números sean consecutivos. Lo más conveniente es numerar las líneas de diez en diez –o incluso más– de forma que siempre quede la posibilidad de introducir nuevas líneas con números intermedios.

Los ejercicios de los próximos epígrafes aclararán suficientemente este punto.

El segundo y **el tercer campos** conforman el cuerpo de la sentencia y son el verdadero

objeto de este CURSO. No obstante, podemos anticipar que *el segundo campo* estará ocupado siempre por alguna de las palabras del vocabulario BASIC, y *el tercer campo* podrá estar ocupado por una **variable**, una **constante,** una **expresión matemática** o una **cadena de caracteres.**

Entendemos por cadena de caracteres cualquier combinación de letras, números y símbolos.

El cuarto campo está destinado a indicar al ordenador que la línea ha terminado y, por tanto, debe prepararse para recibir otra nueva. El comando destinado a este fin es **ENTER.**

EJEMPLO:

```
1Ø INPUT A$
2Ø PRINT A$
```

Este sencillo programa está compuesto por dos sentencias:

- La primera está en la línea de programa número 1Ø y ordena al computador que espere la entrada (INPUT) de una *variable alfanumérica* denominada A$.
- La segunda línea tiene el número 2Ø y ordena al computador que imprima en pantalla el contenido de la variable A$.

Recuerde que después de escribir cada una de las líneas anteriores debe teclear ENTER.

Esta palabra se deriva de MUSA AL-KHOWARIZMI, nombre de un matemático árabe que vivió en el siglo IX.

El diccionario define la palabra **algoritmo** como "método y notación en las distintas formas de cálculo."

Un *algoritmo* es, en definitiva, un conjunto de operaciones perfectamente especificadas cuyo objeto es obtener un resultado partiendo de unos datos.

EJEMPLO:

¿Cuál es el algoritmo que nos permite conocer la superficie de un cuadrado en función de la longitud de su lado?

SOLUCION:

S: superficie del cuadrado.

l: lado del cuadrado.

Algoritmo buscado: **S = l * l**

En este caso, el dato necesario es la longitud del lado del cuadrado y el resultado que se obtendrá es la superficie de esta figura geométrica.

El conjunto de operaciones de este algoritmo se reduce a multiplicar el valor de **l** por sí mismo.

El algoritmo *S = l * l* es válido para cualquier valor de *l* y, en general, de las variables que intervienen.

2-3 CONSTANTES

Todos los valores que permanecen invariables en un algoritmo se llaman **constantes.**

EJEMPLO:

S: superficie del triángulo.
b: base del triángulo.
h: altura del triángulo.

$$S = \frac{b * h}{2}$$

En este algoritmo, que determina la superficie de un triángulo, tenemos *la constante* 1/2.

Hay dos tipos de constantes: **alfanuméricas** y **numéricas.**

Las constantes alfanuméricas están formadas por cualquier combinación de caracteres, siempre que estén escritos entre comillas. Por ejemplo, "SEAT 127" ó "1, 2, 3... Responda otra vez".

Las constantes numéricas son números positivos o negativos, enteros o decimales, y no van entre comillas. La coma decimal se representa por un punto (.).

Existen dos tipos de *constantes numéricas:*

A. *Enteras*

Están formadas por cualquier número, positivo o negativo, y no pueden tener decimales. Por ejemplo, 21729 ó –18617.

B. *Reales*

Una constante real es cualquier elemento del conjunto de los números reales. Es decir, abarca todos los números positivos y negativos, enteros y decimales. Por ejemplo, 217, –3142, 2.147 ó –34.4157.

VARIABLES 2-4

Una **variable** representa un valor que *puede cambiar* a lo largo de un algoritmo, pero que *permanece fijo* mientras no haya una instrucción que indique lo contrario. *Las variables* también pueden ser **numéricas** y **alfanuméricas.**

Las variables numéricas se representan por cualquier conjunto de caracteres, con la única condición de que el primero sea una letra. Por ejemplo, LET a = 5; LET A = 7; LET AB = 15, o LET AB1 = 45.

Las variables alfanuméricas o *de caracteres* se representan por una sola letra, mayúscula o minúscula, seguida del símbolo $. De esta forma, el computador distingue que el contenido de esa variable es distinto de un número y lo interpretará como una cadena de caracteres. Esta cadena debe estar entrecomillada. Por ejemplo, LET a$ = "ordenador", o LET P$ = = "país".

Se entiende por **nombre de la variable** cualquier combinación de caracteres –de acuerdo con la sintaxis anteriormente explicada– que la represente. En los ejemplos anteriores, hemos denominado **a, A, AB** y **AB1** las variables numéricas, y **a$** y **P$** las variables alfanuméricas.

2-5	EXPRESIONES

Anteriormente nos hemos referido a las variables como elementos capaces de contener un valor. Una **expresión,** por el contrario, **es un valor.**

Nada impide, sin embargo, que el valor de una expresión varíe de acuerdo con los valores de las variables que forman parte de la expresión.

EJEMPLO:

Definir una expresión que determine el valor del importe de una conferencia a Sevilla, sabiendo que cada "paso" cuesta 10 ptas.

SOLUCION:

Importe de la conferencia = **10 * n.**

En esta expresión **10 * n** tenemos:

- Una constante numérica **10**, que es el precio de un "paso".
- Una variable numérica **n**, que representa el número de "pasos".

Evidentemente, para cada valor de n tendremos un valor diferente de la expresión, que nos dará los diferentes importes de las conferencias a Sevilla, según su número de pasos.

Las expresiones matemáticas se escriben en BASIC prácticamente igual que con lápiz y papel, pero con las variaciones impuestas por los símbolos que figuran en el teclado del ordenador. Tal es el caso, por ejemplo, de x^3 (equis elevado a 3), que nos obligará a teclear **x ↑ 3.**

El orden de prelación de las diferentes operaciones matemáticas se verá más tarde, pero, antes de llegar al mismo, es importante hacer observar unas sutiles diferencias que existen entre escribir una expresión matemática a mano o en un computador. Supongamos que deseamos transcribir la expresión $\frac{a + b}{2\,c}$ de forma entendible para el ordenador. Para lograrlo, no debemos olvidar ningún signo, como * entre 2 y c en el denominador, y no magnificar la capacidad de la máquina, ya que:

- Si escribimos **a + b/2 * c**, el ordenador interpretará $a + \frac{b}{2} * c$.
- Si escribimos **(a + b)/2 * c**, interpretará $\frac{a + b}{2} * c$.
- Sólo podrá interpretar correctamente la expresión citada si escribimos **(a + b)/(2 * c).**

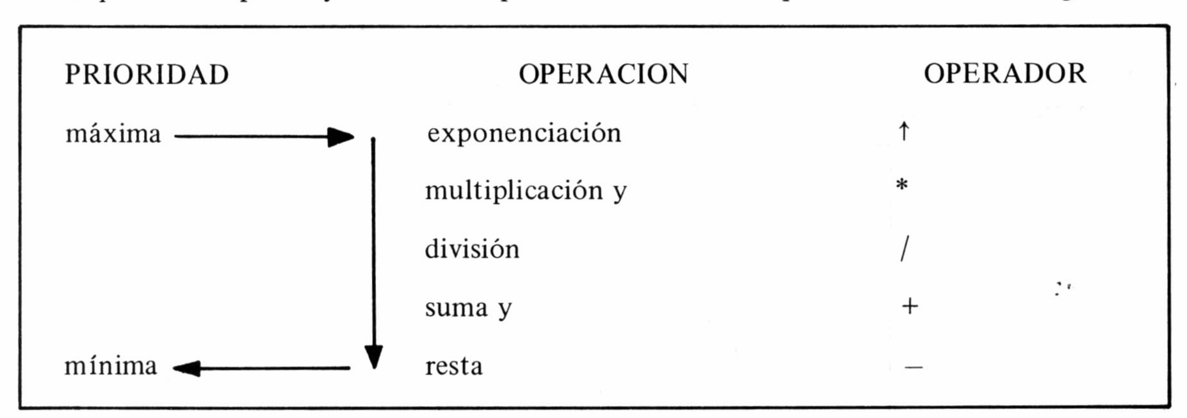
Los **operadores** son símbolos que representan el conjunto de todas **las operaciones –aritméticas** o **lógicas**– que se pueden realizar entre valores.

Estos valores pueden ser –obviamente–, a su vez, expresiones. Son **los operandos.**

Como puede observarse en el ESQUEMA DE COMANDOS BASIC, se dispone de cuatro tipos de operadores: *aritméticos, de relación, lógicos* y *funcionales.*

OPERADORES ARITMETICOS

Se denominan así porque actúan entre valores aritméticos, siendo el conjunto de los símbolos que los componen y su orden de prioridad establecidos por el ordenador los siguientes:

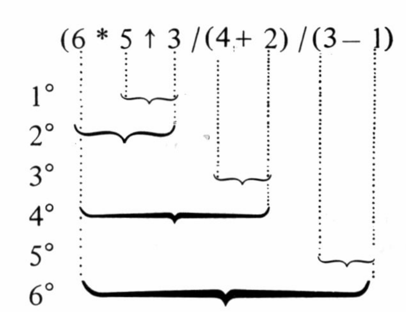


PRIORIDAD	OPERACION	OPERADOR
máxima →	exponenciación	↑
	multiplicación y	*
	división	/
	suma y	+
mínima ←	resta	–

Para operadores de la misma prioridad, la máquina ejecuta las operaciones de izquierda a derecha.

Las operaciones dentro de un paréntesis las efectúa primero y siguiendo el orden de prioridad anterior.

EJEMPLO:



Calcula la potencia 5 ↑ 3 (5^3)
Multiplica el resultado por 6.
Calcula 4 + 2
Divide el producto (2°) entre la suma (3°)
Calcula 3 – 1
Divide el cociente (4°) entre la diferencia (6°).

Las cadenas de caracteres sólo pueden sumarse.

La forma de yuxtaponer una cadena de caracteres a otra es intercalar entre ellas el operador +.

EJEMPLO:

PROGRAMA	COMENTARIO
1Ø LET A$ = "ABC"	Con las líneas 1Ø y 2Ø asignamos valores a las variables de caracteres A$ y B$.
2Ø LET B$ = "DEF"	
3Ø PRINT A$ + B$	Con la línea 3Ø ordenamos la impresión del contenido actual de A$ y B$, justamente uno a continuación del otro.

El resultado de ejecutar este programa (¡recuerde: RUN!) será:

ABCDEF

OPERADORES DE RELACION

Estos operadores sólo actúan en proposiciones entre dos operandos, cuyo resultado únicamente puede ser **CIERTO**, que se representará por 1, o **FALSO**, que se representará por Ø.

Sus símbolos son:

igual que . . . . =
distinto que . . . . < >
menor que . . . . <
mayor que . . . . >
menor o igual que . . . . < =
mayor o igual que . . . . > =

Si un operador de relación compara dos expresiones en las que intervienen operadores aritméticos, antes de efectuar la comparación, actúan los operadores aritméticos.

EJEMPLO:

Determinar si 2 + 5 es menor que $(15 + 4)^3$ /20250

SOLUCION:

Es evidente que, sin efectuar las operaciones aritméticas que determinan el valor de cada expresión, es difícil responder a la proposición anterior.

Las cadenas de caracteres pueden ser comparadas con estos mismos operadores. Para hacerlo se cotejan, carácter a carácter, ambas cadenas valorando cada carácter de acuerdo con su **código ASCII.** Más adelante se estudiará todo lo referente a los códigos, baste por ahora saber que cada carácter tiene su propio valor.

OPERADORES LOGICOS

Responden directamente a las **funciones lógicas del álgebra de BOOLE.**

Los operadores lógicos actúan entre operandos en los que, a su vez, intervienen operadores de relación. Esto quiere decir que los operandos de los operadores lógicos tendrán siempre el valor **1** (CIERTO) ó **∅** (FALSO), y, por consiguiente, *los operadores lógicos* sólo responden **1** ó **∅** (CIERTO o FALSO).

Lòs operadores lógicos, según su orden de prioridad, son **NOT, AND** y **OR**, que estudiaremos con detalle más adelante.

Ver apéndice FUNCIONES LOGICAS APLICADAS.

OPERADORES FUNCIONALES

Estos son algoritmos que residen en el propio sistema y operan sobre datos suministrados al computador por medio del teclado o el programa, obteniendose como resultado un valor.

Las funciones de este tipo pueden ser numéricas, alfanuméricas y operativas.

En el ESQUEMA DE COMANDOS BASIC se puede ver el conjunto de estos operadores.

2-7 TECLEANDO UN PROGRAMA

Al estudiar la estructura de una línea de programa, vimos que el cuarto campo está dedicado a indicar al computador el FIN DE LINEA. Esto quiere decir que, una vez tecleada la línea completa, debemos apretar **ENTER** para ordenar a la máquina que la memorice, cosa que hará si no hay ningún error de sintaxis.

Supongamos que la línea que vamos a introducir es: **152Ø INPUT A$**

En primer lugar, teclearemos el número de línea: **152Ø**; a continuación, el comando **INPUT**, y, finalmente, los caracteres **A** y **$**. Con todo esto la línea habrá concluido y sólo faltará indicar al computador que la pase a su memoria, para lo cual apretaremos la tecla **ENTER.**

Todos los computadores tienen un símbolo –normalmente >– para indicar la última línea memorizada, que se sitúa automáticamente entre el número de línea y la sentencia. Este indicador se llama **puntero.**

Si la línea no es aceptada por el ordenador debido a algún tipo de error, deberemos proceder a corregirlo.

En primer lugar, debemos observar que, en la zona de la pantalla donde van apareciendo los diferentes caracteres a medida que se teclean, un **cursor** se va desplazando y queda siempre a la derecha del último carácter tecleado.

Este *cursor* está representado por diferentes símbolos. Todo lo referente a este tema se estudia en el Apéndice A.

El cursor intermitente indica dónde está situado, y la forma en que queda representado en la pantalla no importa tanto como el hecho de saberlo desplazar, voluntariamente, a derecha e izquierda a lo largo de la línea que estemos manipulando.

Tanto el *puntero* como el *cursor* se desplazan mediante las teclas donde figuran las "flechas" (↑ ↓ ← →) que indican el sentido deseado. En el "plus" basta con apretar dichas teclas y en el "convencional" hay que apretar además CAPS SHIFT.

Visto esto, hay que desplazar *el cursor* justo a la derecha del carácter equivocado y borrarlo apretando la tecla DELETE.

Supongamos que en la línea **1Ø INPUT A$**

queremos sustituir **A** por **B**. Una vez situado *el cursor* a la derecha de **A**, hay que apretar la tecla DELETE – o CAPS SHIFT y DELETE en el "convencional"–, con lo que se borrará **A** y sólo restará escribir **B** en su lugar.

Hay que destacar que no hay nada que impida, una vez posicionado *el cursor* en el lugar adecuado, insertar nuevos elementos en la línea.

Todos estos detalles se verán con más claridad a medida que avancemos en la lectura de esta sección. Sin embargo, podemos anticipar que, si la línea de programa que queremos corregir está ya memorizada por el computador, podemos manipularla de acuerdo con las indicaciones que, para tal fin, existen. Este proceso se llama **"editar la línea".**

ESQUEMA DE COMANDOS BASIC

Operadores aritméticos: ↑, *, /, +, –, ( )

Operadores de relación: =, <,>, < >, <=, >=

Operadores lógicos: NOT, AND, OR

Operadores funcionales:

- *Operativos:* *ATTR, IN, PEEK, OUT, POKE*
- *Alfanuméricos:* CHR$, CODE o ASC, INKEY$, LEN, STR$, VAL, VAL$, ABS, ACS, ASN, ATN
- *Numéricos:* COS, EXP, INT, LN ó LOG, RND, SGN, SIN, SQR, TAN

Sentencias:

- *Auxiliares* *BEEP, BORDER, BRIGHT, CIRCLE,* CLEAR, CLS, CONTINUE, COPY, *DRAW, FLASH, INK, INVERSE,* LIST, LLIST, LOAD, LPRINT, MERGE, NEW, *OVER, PAPER,* PAUSE, *PLOT,* PRINT, REM, RUN, SAVE, STOP, VERIFY
- *De asignación* DATA, DEF FN, DIM, INPUT, LET, RANDOMIZE, READ, RESTORE
- *De bifurcación incondicional:* GO SUB, GO TO, RETURN
- *De selección:* IF/THEM
- *De repetición:* FOR ... TO/NEXT, FOR ... TO ... STEP/NEXT

NOTA: Los comandos en cursiva se explican detalladamente en la Sección III.

2-9	CONFIGURACION DE LOS CAPITULOS

Cada ficha responde a la siguiente estructura:

COMANDO	SINTAXIS	SIGNIFICADO

INTERPRETACION	Explica el uso general de la palabra.
POSIBILIDADES	Explica las diferentes posibilidades de utilización.
FORMA de teclear la instrucción:	
EJEMPLOS	Para aclarar el campo de utilización.

Para situar los comandos ver **APENDICE A**, "CROQUIS DE ACCESO RAPIDO A LOS COMANDOS".

ABS	ABS (expresión numérica)	ABSOLUTO

INTERPRETACION:

Se obtiene el valor absoluto de la expresión numérica.

POSIBILIDADES:

La expresión numérica puede ser un número o una expresión numérica que, como ya es sabido, finalmente es un número.

FORMA de teclear la instrucción: **Apretar G en modo E**

EJEMPLO:

PROGRAMA	COMENTARIOS
PRINT ABS (–5)	Si tecleamos esta instrucción y apretamos ENTER obtenemos el valor absoluto de –5, que es 5.

> Cuando se teclea una instrucción sin línea de programa, esperando respuesta inmediata del ordenador, estamos trabajando en modo directo.

EJEMPLO:

PROGRAMA	COMENTARIOS
1Ø LET a = 25 **2Ø LET b = 1Ø** **3Ø PRINT ABS (25 * 1Ø)**	En este caso, la respuesta del ordenador será 25Ø.

CHR$	CHR$ (expresión numérica)	CARACTER

INTERPRETACION:

Se obtiene el carácter cuyo código es la expresión numérica indicada.

POSIBILIDADES:

Los valores a asignar en la expresión numérica deben estar comprendidos entre Ø y 255, ambos inclusive, que corresponden **al código ASCII de caracteres.**

FORMA de teclear la instrucción: **Apretar U en modo E**

EJEMPLO:

CHR$ 65 corresponde al carácter **A.** Para comprobarlo, basta teclear PRINT CHR$ 65 seguido de ENTER.

INTERPRETACION:

Se obtiene el código ASCII del primer carácter de la expresión alfanumérica.

POSIBILIDADES:

Es la función inversa de CHR$.

FORMA de teclear la instrucción: **Apretar I en modo E**

EJEMPLO:

CODE "Angel" nos retornará **65**, que es el código de A, primer carácter de la expresión alfanumérica "Angel". Para comprobarlo, basta teclear PRINT CODE "Angel" seguido de ENTER.

COPY	COPY	COPIA

INTERPRETACION:

Copia en papel – mediante impresora– el contenido exacto de la pantalla.

POSIBILIDADES:

COPY

FORMA de teclear la instrucción: **Apretar Z en modo K**

COS	COS (expresión numérica)	COSENO

INTERPRETACION:

Se obtiene el valor del coseno de la expresión numérica, dado el ángulo en radianes.

POSIBILIDADES:

Las que se derivan del cálculo trigonométrico.

FORMA de teclear la instrucción: **Apretar W en modo E**

EJEMPLO:

Escribir el programa que, al ejecutarlo, imprima en pantalla el valor de los cosenos de los ángulos de 0,5 en 0,5 radianes, entre 0 y 2 π.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø FOR X = Ø TO 2 * PI STEP 0.5 **2Ø PRINT COS X** **3Ø NEXT X**	Si el computador en uso dispone de otras funciones trigonométricas, los criterios de utilización de las mismas serán similares a los expuestos hasta aquí.

DATA/READ		DATOS/LEER

DATA	DATA constantes	DATOS

INTERPRETACION:

Con esta sentencia se almacenan valores –numéricos o alfanuméricos– a los cuales accede el programa secuencialmente, gracias a la sentencia **READ**, que estudiaremos después.

POSIBILIDADES:

Cada constante, dentro de la sentencia DATA, debe ir separada de la siguiente por una coma, y las constantes alfanuméricas deben escribirse entre comillas.

Las sentencias DATA son interpretadas por el BASIC como **almacén de datos** y, consiguientemente, no implican ningún tipo de operación al ser leídas. Esto quiere decir que las sentencias DATA se pueden colocar en cualquier parte del programa, sin que afecte al desarrollo del mismo; no obstante, es aconsejable colocar todos los DATA en un mismo sector del programa.

Un DATA puede contener tantas constantes como requiera el programador, y un programa puede tener tantos DATA como sean necesarios.

La sentencia DATA, para que sea operativa, necesita de la sentencia READ.

FORMA de teclear la instrucción: **READ: Apretar A en modo E**
DATA: Apreter D en modo E

EJEMPLO:

Almacene en la línea de programa 1ØØØ los nombres de cinco capitales europeas.

SOLUCION:

1ØØØ DATA "Madrid", "París", "Londres", "Roma", "Dublín".

INTERPRETACION:

Con estas sentencias se leen, secuencialmente, las constantes almacenadas en las sentencias DATA, asignando estos valores —uno tras otro— a las variables. Estas variables van separadas por comas.

POSIBILIDADES:

Cuando la lectura de un programa llega a una sentencia READ, lee la primera constante aún no leída de todos los valores almacenados en todas las sentencias DATA del programa, asignando dicho valor a la variable correspondiente.

Las variables deben ser del mismo tipo —numéricas o alfanuméricas— que las constantes que lee de los DATA. En caso contrario, un mensaje de error avisará de la anomalía.

Una sentencia READ puede acceder a uno o varios DATA, o varios READ a un solo DATA, pero siempre secuencialmente.

Si la cantidad de variables situada tras una sentencia READ es mayor que el total de constantes, saldrá en la pantalla un mensaje de error advirtiendo que no hay suficientes constantes. Este mensaje suele ser OUT OF DATA, indicando a continuación el correspondiente número de línea.

Si hay más constantes en las sentencias DATA que variables en las sentencias READ, el excedente de constantes queda guardado hasta que otras sentencias READ obliguen a utilizarlo. Es decir, las variables de la siguiente sentencia READ tomarán los valores aún no leídos del total de las constantes de las sentencias DATA.

Para comenzar a leer las constantes desde el primer DATA hay que usar la sentencia *RESTORE*, como veremos posteriormente.

FORMA de teclear la instrucción: **READ: Apretar A en modo E**

EJEMPLO:

1. Escriba el programa, que una vez ejecutado, imprima en pantalla las cinco ciudades europeas del ejemplo anterior.

SOLUCION 1:

PROGRAMA	COMENTARIOS
1Ø READ A$, B$, C$, D$, E$	En la línea 1Ø damos valores a las variables indicadas, extrayéndolas secuencialmente del DATA de la línea 1ØØØ.
2Ø PRINT A$, B$, C$, D$, E$	En la línea 2Ø se ordena su impresión.
1ØØØ DATA "Madrid", "París", "Londres", "Roma", "Dublín"	

SOLUCION 2:

PROGRAMA	COMENTARIOS
1Ø FOR X = 1 TO 5 **2Ø READ V$** **3Ø PRINT V$** **4Ø NEXT X** **1ØØØ DATA "Madrid", "París"** **1Ø1Ø DATA "Londres", "Roma", "Dublín"**	En esta ocasión, una sola variable V$ va tomando sucesivamente los valores contenidos en los DATA.

EJEMPLO:

2. Escriba el programa que, una vez ejecutado, imprima en pantalla el nombre de cinco alumnos y sus calificaciones.

SOLUCION:

PROGRAMA	**COMENTARIOS**
1Ø FOR X = 1 TO 5 **2Ø READ V$: PRINT V$** **3Ø READ V: PRINT V** **4Ø NEXT X** **1ØØ DATA "JUAN", 5, "José", 6, "Ricardo", 4, "Luis", 8, "Tomás", 9.**	Observe cómo la secuencia de lectura de los READ exige que los valores de los DATA sean, alternativamente, alfanuméricos y numéricos.

EJEMPLO:

3. Escriba el programa que, una vez ejecutado, imprima en pantalla la suma de los números pares y la suma de los números impares menores que 1Ø.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø DATA 2, 4, 6, 8, 1, 3, 5, 7, 9, "pares < 1Ø", "impares < 1Ø"	Las sentencias READ de las líneas 2Ø, 3Ø y 5Ø van tomando los valores que hay en la sentencia

PROGRAMA	COMENTARIOS
2Ø READ A, B, C, D **3Ø READ E, F, G, H, I** **4Ø LET S = A + B + C + D** **5Ø LET T = E + F + G + H + I** **6Ø READ A$, B$** **7Ø PRINT "La suma de los números "; A$; " es "; S; " y la suma de los números "; B$; " es "; T**	DATA de la línea 1Ø. Observe que las variables A$ y B$ de la línea 6Ø, que son alfanuméricas, toman los valores escritos entre comillas en la línea 1Ø. Recuerde: No hay problema cuando hay más constantes que variables. Las constantes que sobran quedan a la espera de que otra u otras sentencias READ los ordene leer.

DEF FN	DEF FN variable = expresión	DEFINIR FUNCION

INTERPRETACION:

Con esta instrucción, el usuario del ordenador puede definir sus propias funciones, siguiendo las reglas que se explican a continuación y de forma análoga al proceso mental matemático de decir: **"La función y de x – y (x) – es igual a tal expresión."**

POSIBILIDADES:

Las funciones numéricas se reconocen y se denominan mediante instrucciones **FN** seguidas de una sola letra. Por ejemplo, **DEF FN A.**

FORMA de teclear la instrucción: **Apretar 1 SYMBOL SHIFT en modo E**

EJEMPLO:

Para definir una función en BASIC se debe proceder de una forma similar a ésta:

```
10 DEF FN A (x) = x + x * 2
```

La función aquí definida nos dice que, para un valor de **x**, la función **A (x)** es igual a la expresión indicada. Dicho esto, cabe preguntarse cómo se dan valores a la variable **x**. Lo veremos a continuación al tratar **el comando FN.**

En el caso de expresiones alfanuméricas podría ser, por ejemplo:

```
10 DEF FN A$ (B$) = B$ (3 TO 10)
```

Aquí se define una cadena de caracteres (A$) en función de otra (B$).

DIM	DIM. nombre de la matriz (definición de elementos)	DIMENSIONAR

INTERPRETACION:

Una matriz en BASIC debe ser concebida como un archivo clasificado de elementos. Estos elementos pueden ser números o caracteres.

Con la sentencia **DIM** se dimensiona **una matriz** denominada "nombre de la matriz" y organizada según las especificaciones dadas en la definición de elementos, reservándose la memoria suficiente para almacenar todos sus elementos. Es decir, **se establecen las dimensiones de esa matriz.**

POSIBILIDADES:

El nombre de la matriz sólo puede ser una letra y la definición de elementos incluye diferentes conceptos, según el tipo de matriz que se pretenda dimensionar. Estos tipos dependerán de la clase de elementos –números o caracteres– y de la organización de los elementos dentro de la matriz.

En una misma matriz sólo puede haber elementos numéricos o, por el contrario, alfanuméricos.

DIM debe ser introducida en el programa antes de pretender utilizar las posibilidades que ofrecen las matrices en BASIC y que estudiamos a continuación.

Para el dimensionado de matrices alfanuméricas –de caracteres–, el nombre de la matriz debe ser una letra seguida del símbolo $. A las matrices numéricas les basta simplemente con la letra.

Los elementos de una matriz pueden organizarse según **una sola dimensión** –secuencia lineal de elementos– o **multidimensionalmente**, distribuyéndolos en filas y columnas e, incluso, más allá como tendremos oportunidad de ver.

Los elementos de cualquier matriz en BASIC son, a todos los efectos, variables cuyo nombre genérico es el mismo para todos y coincide con el nombre de la matriz, distinguiéndose unos de otros por la posición que ocupan dentro de ella.

En el momento de dimensionar una matriz –aplicando la sentencia DIM–, todos los elementos de la misma son Ø. O, dicho de otro modo, las variables que conforman la matriz contienen el valor Ø.

A los elementos de una matriz, también se les conoce por **variables con subíndice**, siendo **el subíndice** el indicador de la posición que ocupan en ella.

Las variables con subíndice pueden manipularse en un programa de la misma forma, y sin ninguna restricción, en que han sido utilizadas las variables en general a lo largo de todo este libro.

Vistos estos conceptos generales, pasemos a unos ejemplos para precisar las ideas.

En este sentido es conveniente aclarar que los ejemplos que se plantean a continuación parten del supuesto de **la necesidad de utilizar matrices** y, consiguientemente, la aplicación de las mismas es inmediata. De aquí podrá el lector sacar conclusiones de tipo operativo que le son imprescindibles, pero –y esto es lo más importante– para usar las matrices en BASIC de una forma eficiente, el programador debe plantearse con claridad el porqué de la matriz que piensa definir.

FORMA de teclear la instrucción: **Apretar D en modo K**

Teclear **DIM, el nombre de la matriz y la definición de los elementos separados por comas y entre paréntesis**, seguido de **ENTER**.

EJEMPLO:

Introducción al planteamiento de matrices numéricas.

Supongamos que es de nuestro interes tener controladas las notas obtenidas por los alumnos de tres centros escolares con cinco cursos de diez alumnos cada centro y, de momento, no es necesario conocer sus nombres, sino sólo sus notas para llegar a conclusiones de tipo estadístico.

En primer lugar, asociaremos uno de los centros al número 1; dentro de ese centro vamos a denominar 1 también al primer curso, 2 al segundo, 3 al tercero, 4 al cuarto y 5 al quinto, y, dentro de cada curso, volvemos a dar al número 1 el significado de primer alumno, al 2 el de segundo alumno, y así, sucesivamente, hasta llegar al décimo con el número 10.

Todo esto lo habremos hecho con la intención de poder saber que, si nos encontramos con una información del tipo (1, 4, 8), estamos refiriéndonos al primer centro, cuarto curso y octavo alumno.

Con el mismo criterio (2, 1, 2) significará que es el segundo centro del primer curso del segundo alumno. Y un (3, 2, 10) vendrá a decirnos que estamos en el tercer centro, segundo curso, décimo alumno.

Gracias a este planteamiento, hemos estructurado los subíndices de nuestra futura matriz de forma que signifique algo congruente para el programador.

Razonamientos de este tipo intuitivo o altamente reflexionado, según la complejidad del asunto, deben ser seguidos para que la organización de los elementos dentro de la matriz responda al criterio de eficacia que se debe exigir.

Bien, una vez en este punto, es relativamente fácil llegar a hacer operativa una matriz.

Comenzaremos por dimensionar la matriz, para la cual debemos darle al ordenador datos que la ponderen. Así, en nuestro ejemplo, con **DIM N** estamos ordenando al computador que dimensione una matriz numérica cuyo nombre sea **N**, pero ahora necesita saber cuántos elementos la van a componer y cómo van a estar distribuidos.

La instrucción completa será **DIM N (3, 5, 1Ø)** –3 centros, 5 cursos por centro, 10 alumnos por curso–, la cual, una vez ejecutada, habrá originado en el computador la creación de **150 variables**, cuyo contenido inicial es **Ø**, de nombre N y con subíndices que van desde (1, 1, 1) hasta (3, 5, 10).

De ahora en adelante, para referirnos a un elemento cualquiera de esta matriz, lo haremos mediante la expresión **N (C, R, A)**, de forma que esta notación viene a decir nota del **alumno** A, del **curso R**, del **centro C.**

De forma explícita, las variables de las que disponemos en memoria son:

Correspondientes al primer centro

N (1, 1, 1) N (1, 1, 2) N (1, 1, 3) N (1, 1, 4) N (1, 1, 5) N (1, 1, 6)
N (1, 1, 7) N (1, 1, 8) N (1, 1, 9) N (1, 1, 10)
N (1, 2, 1) N (1, 2, 2) N (1, 2, 3) N (1, 2, 4) N (1, 2, 5) ..... N (1, 2, 10)
N (1, 3, 1) N (1, 3, 2) N (1, 3, 3) .... N (1, 3, 10)
N (1, 4, 1) N (1, 4, 2) N (1, 4, 3) .... N (1, 4, 10)
N (1, 5, 1) N (1, 5, 2) N (1, 5, 3) .... N (1, 5, 10)

Correspondientes al segundo centro	N (2, 1, 1)	N (2, 1, 2)	N (2, 1, 3)	. . . .	N (2, 1, 10)
	N (2, 2, 1)	N (2, 2, 2)	N (2, 2, 3)	. . . .	N (2, 2, 10)
	N (2, 3, 1)	N (2, 3, 2)	N (2, 3, 3)	. . . .	N (2, 3, 10)
	N (2, 4, 1)	N (2, 4, 2)	N (2, 4, 3)	. . . .	N (2, 4, 10)
	N (2, 5, 1)	N (2, 5, 2)	N (2, 5, 3)	. . . .	N (2, 5, 10)
Correspondientes al tercer centro	N (3, 1, 1)	N (3, 1, 2)	N (3, 1, 3)	. . . .	N (3, 1, 10)
	N (3, 2, 1)	N (3, 2, 2)	N (3, 2, 3)	. . . .	N (3, 2, 10)
	N (3, 3, 1)	N (3, 3, 2)	N (3, 3, 3)	. . . .	N (3, 3, 10)
	N (3, 4, 1)	N (3, 4, 2)	N (3, 4, 3)	. . . .	N (3, 4, 10)
	N (3, 5, 1)	N (3, 5, 2)	N (3, 5, 3)	. . . .	N (3, 5, 10)

Hemos acordado que cualquiera de estas variables está representada por N (C, R, A), lo cual significa que, si hacemos C = 1, R = 2 y A = 4, nos estamos refiriendo a la variable N (1, 2, 4), que figura recuadrada en el esquema anterior y cuyo contenido inicial es Ø.

Ahora ya somos conscientes de lo que significa la primera línea de nuestro programa:

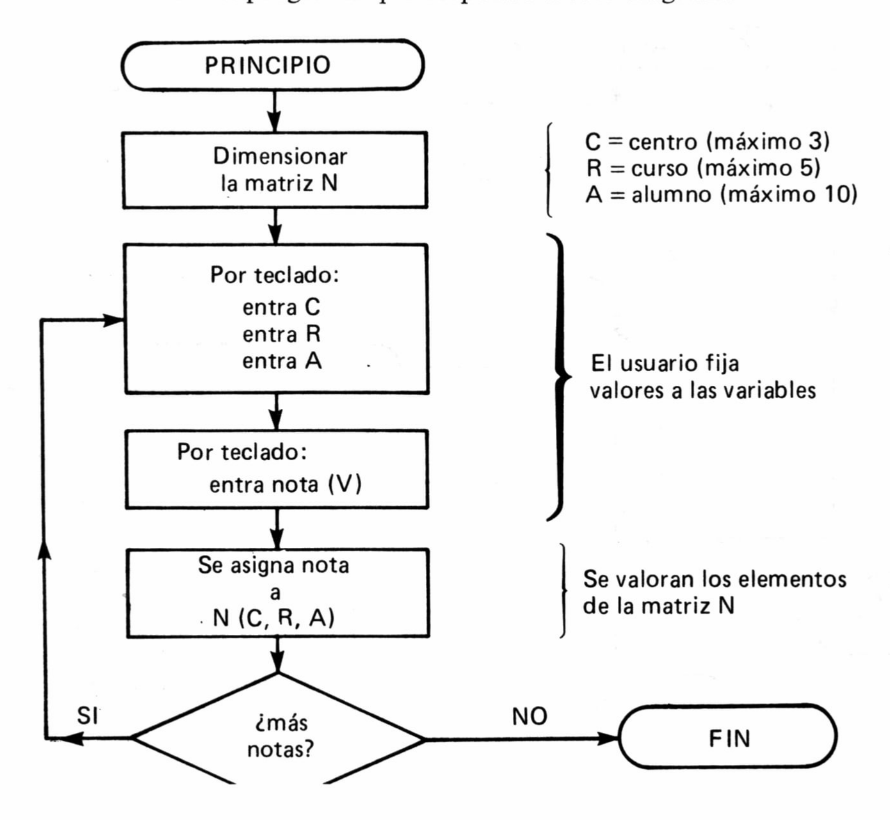
1Ø DIM N (3, 5, 10)

En lo que sigue, haremos operativa esta matriz numérica.

Si quisiéramos darle un valor determinado a un elemento cualquiera de esta matriz, podríamos escribir, por ejemplo,

LET N (1, 2, 4) = 7.5

Pero esta forma de dar valor a las variables de una matriz no suele ser práctica, y, en este caso, no lo es de ningún modo desde el punto de vista del ejemplo propuesto. Resulta más conveniente diseñar un programa que responda a este diagrama:



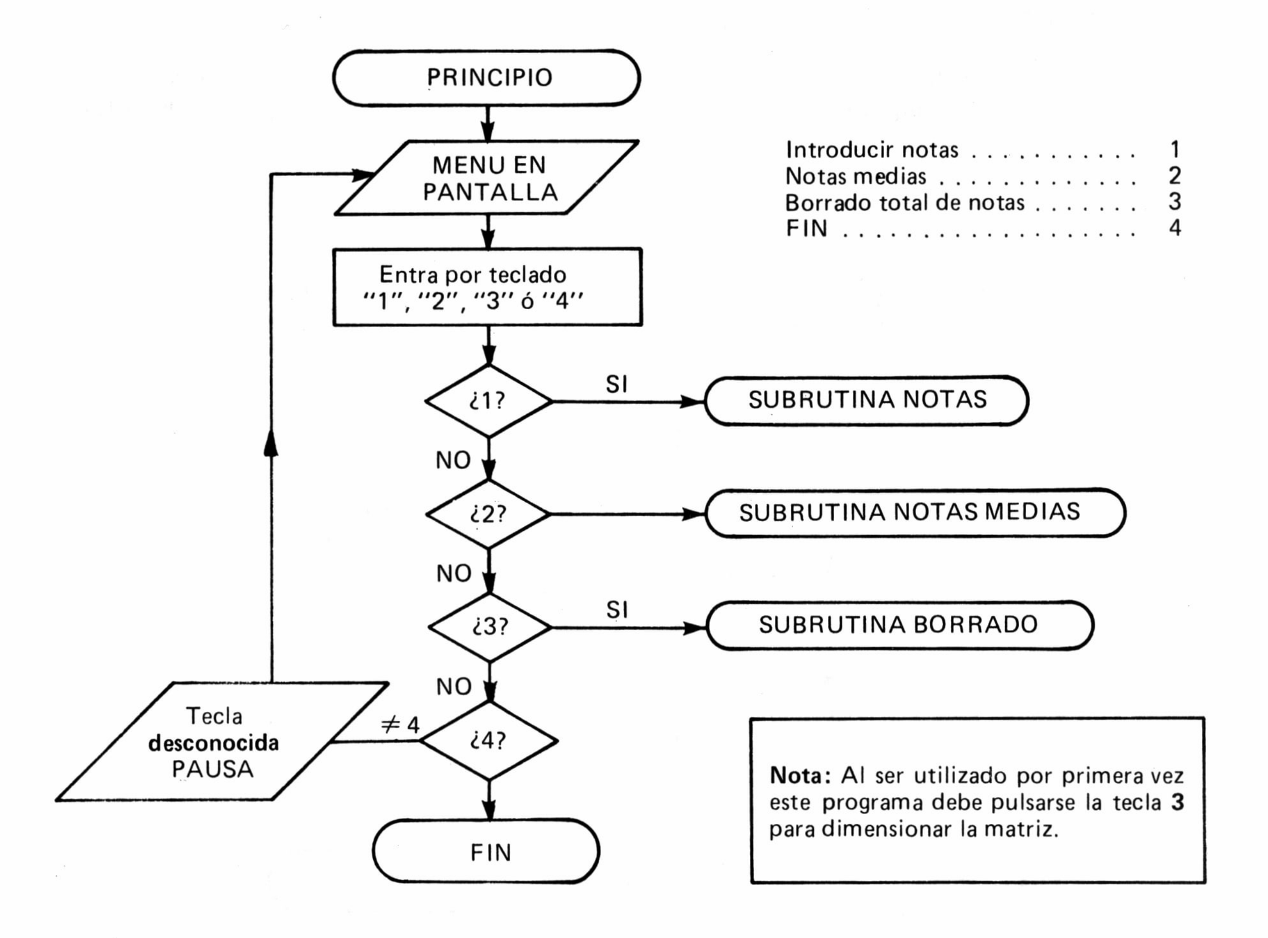
> Un programa con este tipo de diseño no debe ser ejecutado con **RUN/ENTER**, ya que pondría a ∅ el contenido de las variables. Es mejor correrlo con **GO TO** a *la línea siguiente a aquélla donde se dimensiona la matriz.*

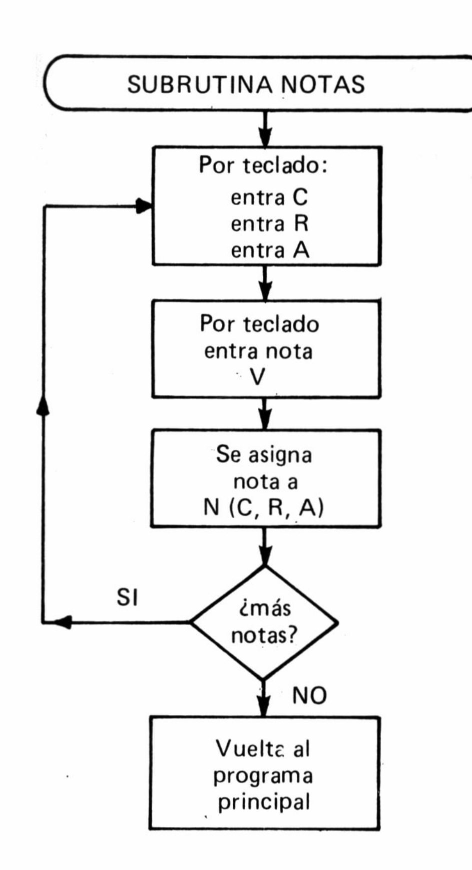
Un listado que corresponda a la idea anterior puede ser:

PROGRAMA	COMENTARIOS
1∅ DIM N (3, 5, 1∅) **2∅ INPUT "Centro 1, 2 ó 3?"; C** **3∅ INPUT "Curso 1, 2, 3, 4 ó 5?";R** **4∅ INPUT "Alumno del 1 al 1∅?";A** **5∅ INPUT "Nota?"; V** **6∅ LET N (C, R, A) = V** **7∅ INPUT "Más notas? (S/N)"; R$** **8∅ IF R$ = "S" THEN GO TO 2∅** **9∅ STOP**	Estudie primero y teclee después este programa. Ejecútelo y dé los valores que quiera a **C, R, A** y **V**. Si suponemos que **C = 1, R = 3, A = 5** y **V = 6**, con el comando directo **PRINT N (1, 3, 5)**, obtendremos un **6** en la pantalla. Recuerde que este programa debe ejecutarlo con GO TO 9, línea anterior a la que dimensiona la **matriz.**

Cuando ejecute este programa y haga sus pruebas, seguramente notará que, si bien la matriz se rellena a medida que da valores a **C, R, A** y **V** —cosa que puede comprobar con el comando directo PRINT seguido de cualquier elemento de la matriz—, no es útil en ningún otro sentido.

Avancemos un poco más con el análisis de los siguientes diagramas:





COMENTARIOS

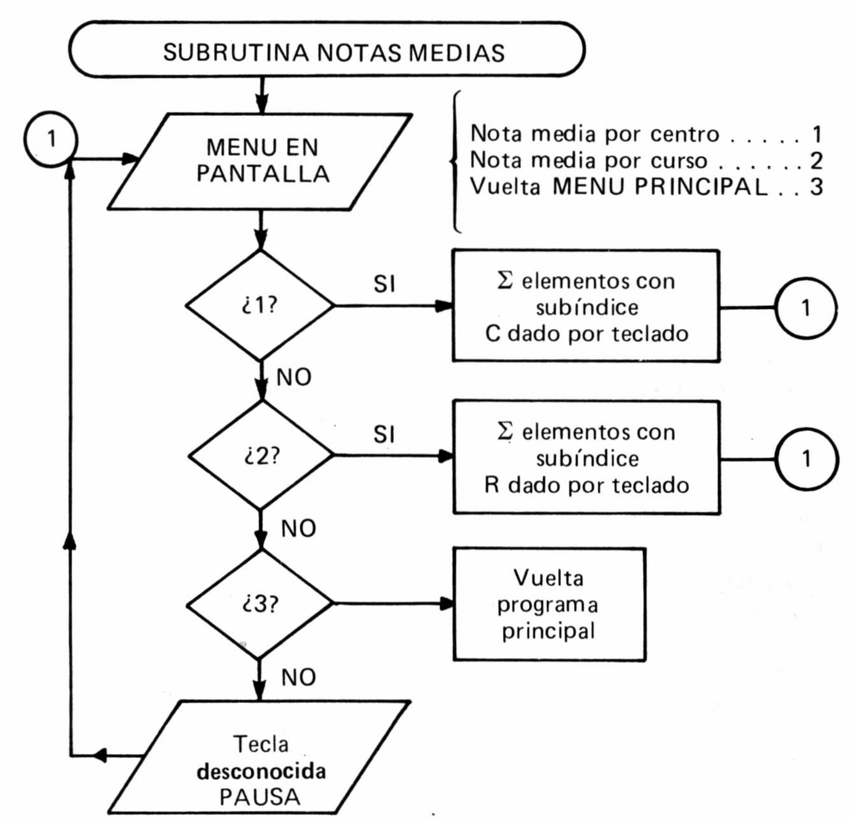
El programa principal, gracias al menú, dirige la secuencia de lectura a una de las tres subrutinas, en función de las necesidades o los deseos del usuario del programa.

Por teclado, entran los valores de las variables C (centro), R (curso) y A (alumno).

También por teclado, entra el valor de la variable V (nota).

Se asigna la nota V al alumno A del curso R del centro C.

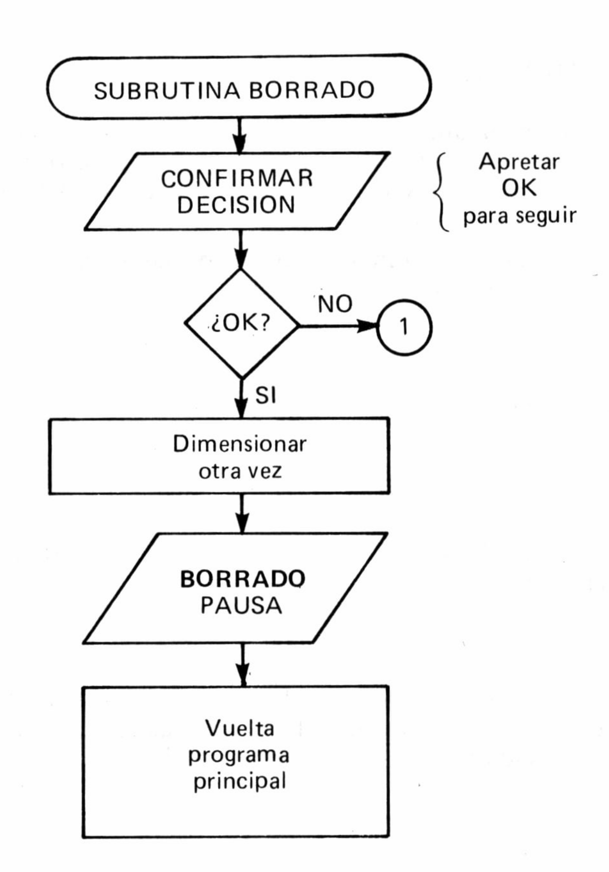
El bucle permite repetir el proceso anterior hasta que se hayan asignado valores a todas las variables o se decida volver al programa principal.



COMENTARIOS

Gracias al menú, el usuario del programa puede obtener la nota media por centro o por curso, o volver al programa principal.

Indica la suma de n elementos con subíndice C o R –introducidos por teclado– dividida por n para obtener la nota media.



COMENTARIOS

Mediante esta subrutina, se confirma la decisión del usuario de borrar los valores de las variables o de volver al programa principal.

Un programa que responda a las características de los diagramas anteriores puede ser el siguiente:

PROGRAMA	COMENTARIOS
	Para una mejor presentación del menú y del uso de PRINT en general, vea la Sección III

```
  1 REM  TO ICHA:" DIM "
 10 PRINT "1 para introducir notas."
 20 PRINT "2 para notas medias."
 30 PRINT "3 para borrado total, o en la primera utilizacion."
 40 PRINT "4 para finalizar el trabajo."
 50 INPUT R$
 60 IF R$="1" THEN  GO SUB 200
 70 IF R$="2" THEN  GO SUB 300
 80 IF R$="3" THEN  GO SUB 700: GO TO 10
 90 IF R$="4" THEN  STOP
100 CLS
110 PRINT "TECLA DESCONOCIDA"
120 FOR X=0 TO 100: NEXT X
130 CLS
140 GO TO 10
190 REM "NOTAS"
```

```
200 CLS
210 INPUT "Centro: ";C
220  INPUT "Curso: ";R
230 INPUT "Alumno?: ";A
240 INPUT "Nota?: ";V
250 LET N(C,R,A)=V
260 INPUT "Mas notas? (S/N): ";R$
270 IF R$="S" OR R$="s" THEN  GO TO 210
280 RETURN
300 REM "NOTAS MEDIAS"
310 CLS
320 PRINT "1 Para nota media por centro."
325 PRINT "2 para nota media por curso."
330 PRINT "3 para volver al menu principal."
340 INPUT R$
350 IF R$="1" THEN  GO TO 420
360 IF R$="2" THEN  GO TO 540
370 IF R$="3"  THEN  RETURN
380 CLS
390 PRINT "TECLA DESCONOCIDA"
400 FOR X=0 TO 100: NEXT X
410 GO TO 310
420 INPUT "Centro a analizar?: ";C
430 LET Z=0
440 FOR R=1 TO 5
450 FOR A=1 TO 10
460 LET S=N(C,R,A)
470 LET Z=Z+S
480 NEXT A
490 NEXT R
500 PRINT ''"La nota media del centro: "'C;" es ";Z/50
510 INPUT "Otra consulta? (S/N): ";R$
520 IF R$="S" OR R$="s" THEN  GO TO 310
530 RETURN
540 INPUT "Curso a analizar?: ";R
550 LET Z=0
560 FOR C=1 TO 3
570 FOR A=1 TO 10
580 LET S=N(C,R,A)
590 LET Z=Z+S
600 NEXT A
610 NEXT C
620 PRINT ''"La nota media de los cursos: "'R;" es ";Z/30
630 GO TO 510
700 REM "BORRADO Y PRIMERA VEZ"
710 CLS
720 PRINT "  Esta opcion del menu solo debeusarla cuando se
    utiliza "
725 PRINT "el pro-grama  por  primera vez, o  paraborrar tod
    as las notas."
```

```
730 INPUT "Si lo confirma teclee O.K.: ";R$
740 IF R$<>"O.K." THEN  CLS : GO TO 10
750 DIM n(3,5,10)
760 PRINT "Matriz inicializada"
770 FOR X=0 TO 10: NEXT X
780 CLS : RETURN
```

En la línea 12∅ se ha generado un bucle de retardo que durará más cuanto mayor sea el valor final de X.

EJEMPLO:

Introducción al planteamiento de matrices alfanuméricas.

En esta ocasión vamos a suponer que nos interesa conocer los nombres de los alumnos de los tres centros anteriores.

Antes de entrar propiamente en la resolución de este ejemplo, es conveniente conocer las peculiaridades de las matrices de caracteres de forma paulatina.

De momento, vamos a centrarnos en colocar dentro de una matriz de caracteres los nombres de los alumnos de un curso de un solo centro. Supongamos que sus nombres son: Juan, Luis, Francisco, Angel, Miguel, Tomás, Federico, Félix, José y Andrés.

Como vemos, son diez cadenas de caracteres que tienen:

4 caracteres la primera: "**Juan**"
4 caracteres la segunda: "**Luis**"
9 caracteres la tercera: "**Francisco**"
5 caracteres la cuarta: "**Angel**"
6 caracteres la quinta: "**Miguel**"
5 caracteres la sexta: "**Tomás**"
8 caracteres la séptima: "**Federico**"
5 caracteres la octava: "**Félix**"
4 caracteres la novena: "**José**"
6 caracteres la décima: "**Andrés**"

Comparando la longitud de todas ellas, llegamos a la conclusión evidente que la cadena más larga es la tercera, "**Francisco**", que tiene **9 caracteres.**

Así, pues, tenemos **10 cadenas** a guardar en una matriz y la longitud de la mayor es de **9 caracteres**. Estos son dos datos necesarios para dimensionar esa matriz. Es decir, si a lo largo del listado de un programa nos encontramos, por ejemplo, con:

... DIM N$ (1∅ , 9)

debemos interpretar que se dimensiona una matriz alfanumérica, denominada **N$**, capaz de contener **10 cadenas** de hasta **9 caracteres** cada una.

Cuando se dimensiona una matriz alfanumérica, se reserva en memoria un espacio para tantos caracteres como resulten de multiplicar el número de cadenas por el número de caracteres que –como máximo– vayan a conformarlas, e, inicialmente, su contenido son **espacios en blanco.**

Para ver esto con claridad, consideremos **A** cadenas de **L** letras o caracteres en general y dimensionemos la matriz adecuada:

```
70 ..................
80 ... DIM N$ (A, L)
90 ..................
```

En el momento de ejecutarse esta sentencia, se guardan **A * L** espacios. Esto quiere decir que, si tecleamos, por ejemplo, **PRINT N$ (n)** –siendo **n** mayor o igual que **1** y menor o igual que **A**– en pantalla no aparecería nada, que es justamente lo que se espera de un espacio. Pero si el comando fuera **PRINT CODE N$ (n)**, obtendríamos **32**, que es el código ASCII del **espacio.**

Con esto, venimos a comprobar que, mientras las matrices numéricas contienen **ceros** inicialmente, las matrices alfanuméricas contienen inicialmente **espacios.**

Si en una matriz se dimensiona **DIM N$ (A, L)**, se introduce una cadena de longitud menor que **L**, entonces, los caracteres sobrantes se mantienen en espacios.

Si **L** es mayor que la longitud de la cadena, se ignoran los caracteres sobrantes.

Volvamos al caso del curso de diez alumnos cuyos nombres conocemos y cabe preguntarse: *¿Cómo se rellena la matriz alfanumérica una vez diseñada?*

Podemos seguir un proceso similar al de las matrices numéricas. Veamos el siguiente listado:

PROGRAMA	COMENTARIOS
1Ø DIM N$ (1Ø, 9)	Dimensionamos la matriz N$ de 10 cadenas de hasta 9 caracteres.
2Ø INPUT "Número del alumno? "; A	Se pide número de referencia del alumno que corresponderá a su posición en la matriz. Por tanto, debe ser **1 = < A = < 10.**
3Ø INPUT "Nombre? "; L$	Los caracteres del nombre deben ser **1 = < L = < 9.**
4Ø LET N$ (A) = L$	A tendrá el valor numérico dado en 2Ø, y L$ será la cadena definida en 3Ø.
5Ø INPUT "Otro nombre? (S/N) "; R$	Con las líneas 5Ø, 6Ø y 7Ø establecemos las condiciones para repetir el bucle si hay más nombres para introducir en la matriz.
6Ø IF R$ = "S" THEN GO TO 2Ø	
7Ø STOP	

Al ejecutar este programa y dar a **A** los valores **1, 2, 3, ... 10** y a **L$ los nombres** de los diez alumnos citados (Juan, Luis, etc.), se conseguirá rellenar la matriz **N$** de acuerdo con nuestros deseos. Para comprobarlo, basta ejecutar un comando directo del tipo **PRINT N$ (2)/ENTER** y obtendremos en pantalla **Luis.** Haga otras pruebas.

Nuestro siguiente paso será controlar los cinco cursos de un centro. Para ello, bastará con dimensionar la matriz de la línea 10 de la siguiente forma:

```
..................
... DIM N$ (5, 1Ø, 9)
..................
```

en el supuesto de que no haya ningún alumno cuyo nombre tenga más de **9** caracteres en

ninguno de los **5** cursos. Con que sólo existiera uno cuyo nombre tuviese, por ejemplo, **12** caracteres, tendríamos que modificar la anterior matriz de la siguiente forma:

. . . DIM N$ (5, 1Ø, 12)

claro está, a no ser que no tuviera ninguna importancia perder los tres últimos caracteres en cuestión.

Con la instrucción de la línea 10, el ordenador ha reservado en la memoria espacio para **5 * 10 * 9** caracteres y, todos ellos, están inicialmente ocupados por **espacios.**

Un programa adaptado a la nueva situación puede ser:

PROGRAMA	COMENTARIOS
1Ø DIM N$ (5, 1Ø, 9)	Dimensiona 5 grupos de 10 cadenas de 9 caracteres cada una.
2Ø INPUT "Curso? "; R	1 =< R =< 5
3Ø INPUT "Número del alumno? "; A	**1 =< A =< 10**
4Ø INPUT "Nombre? "; L$	**1 =< LEN L$ =< 9**
5Ø LET N$ (R, A) = L$	R tendrá el valor numérico dado en 2Ø, y A el dado en 3Ø; L$ está definida en 4Ø.
6Ø INPUT "Otro nombre? (S/N) "; R$	
7Ø IF R$ = "S" THEN GO TO 2Ø	
8Ø STOP	

Ahora estudiamos el caso completo de los tres centros con cinco cursos cada uno de diez alumnos.

La forma de dimensionar la matriz parece evidente en función de lo expuesto hasta aquí. Se trata de tres grupos formados por cinco subgrupos cada uno que tienen diez cadenas de nueve caracteres cada una. Todo ello corresponde a la siguiente instrucción:

. . . DIM N$ (3, 5, 1Ø, 9)

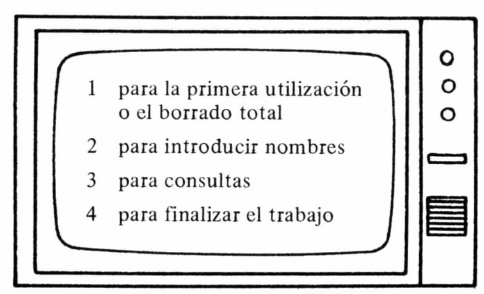
que nos llevaría al siguiente programa:

PROGRAMA	COMENTARIOS
1Ø DIM N$ (3, 5, 1Ø, 9)	Dimensionado de la matriz N$ según lo comentado.
2Ø INPUT "Centro? "; C	**1 =< C =< 3**
3Ø INPUT "Curso? "; R	**1 =< R =< 5**
4Ø INPUT "Número del alumno? "; A	**1 =< A =< 10**
5Ø INPUT "Nombre? "; L$	**1 =< LEN L$ =< 9**
6Ø LET N$ (C, R, A) = L$	C tendrá el valor numérico dado en 2Ø; R el dado en 3Ø, y A el dado en 4Ø; L$ está definida en 5Ø.
7Ø INPUT "Otro nombre? (S/N) "; R$	
8Ø IF R$ = "S" THEN GO TO 2Ø	
9Ø STOP	

Una vez tecleado y ejecutado este programa, damos valores a todas las variables de la matriz hasta rellenar las 3 * 5 * 10 cadenas.

Una vez hecho esto, supongamos que queremos saber el nombre del alumno cuya referencia (matrícula, por ejemplo) es 10, del curso 4 del centro 2. Para ello, podríamos emplear el comando directo **PRINT N$ (2, 4, 10)** y obtendríamos en pantalla el nombre deseado.

Esta forma de averiguar el contenido de los diferentes elementos de la matriz no es muy práctica. Por esta razón, sugerimos al lector el ejercicio de completar el anterior listado de forma parecida a lo establecido en los diagramas de las matrices numéricas, basados en **menús** presentados en pantalla, de los cuales damos una sugerencia en el siguiente esquema:



El lector debe observar que un listado que contenga una combinación adecuada de las instrucciones dadas en el último ejemplo de las matrices numéricas y del que se deriva el comentario anterior, le puede llevar a obtener resultados en los que se unan el nombre del alumno y su nota, utilizando

PRINT N (C, R, A) y **PRINT N$ (C, R, A)**

Este tema, por sí mismo, abriría la posibilidad de una nueva publicación delicada a los ficheros en BASIC, que se sale de los límites que nos hemos propuesto en la presente.

INTERPRETACION:

Se obtiene el valor del número **e** elevado a la potencia representada por la expresión numérica.

FORMA de teclear la instrucción: **Apretar X en modo E**

EJEMPLO:

Elevar el número **e** a la segunda potencia.

SOLUCION:

Teclee **PRINT EXP (2)** y pulse **ENTER**. La pantalla le mostrará el resultado.

INTERPRETACION:

Esta instrucción es el complemento imprescindible de la **DEF FN,** ya que con ella se calcula la *función definida* de acuerdo con el valor dado a la variable involucrada.

POSIBILIDADES:

En trabajos matemáticos o técnicos, estos comandos son muy prácticos. Por esta razón, hemos hecho aquí una referencia a los mismos, pero su campo de posibilidades depende de lo que permita hacer con ellos cada ordenador.

FORMA de teclear la instrucción:

Teclear **FN, el nombre de la variable y el valor de la variable entre paréntesis,** seguido de **ENTER.**

EJEMPLO:

En el caso propuesto en el ejemplo de **DEF FN,** el **FN** funcionaría así para una expresión numérica:

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø DEF FN A (x) = x + x * 2 **2Ø LET y = FN A (7)** **3Ø PRINT y**	En la línea 1Ø se define la función. En la línea 2Ø se da el valor 7 a la variable x y se calcula la expresión definida en 1Ø con este valor.

EJEMPLO:

En el caso de expresiones alfanuméricas tendríamos:

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø LET B$ = "ROMARVELLAVE" **2Ø DEF FN A$ (B$) = B$ (3 TO10)** **3Ø PRINT FN A$ (B$)**	Este tipo de artificio será útil cuando, a lo largo programa, la variable B$ cambie de contenido y sin embargo nos interese una subcadena constante de la misma.

FOR... TO/NEXT FOR... TO... STEP/NEXT		PARA ... HASTA/PROXIMO PARA ... HASTA ... SALTO/PROXIMO

FOR variable = {expresión numérica 1} TO {expresión numérica 2} /NEXT variable

INTERPRETACION:

La variable va cambiando de valor —de unidad en unidad— partiendo del valor dado por la expresión numérica 1, y hasta alcanzar el valor dado por la expresión numérica 2.

El cambio de valor de la variable se producirá —normalmente, de unidad en unidad— cuando en la secuencia normal de lectura se encuentre la sentencia

NEXT variable

Con ella, el programa cambia su secuencia de lectura a la línea donde se encuentra la sentencia inicial FOR ... TO.

POSIBILIDADES:

El cambio de valor de la variable puede ser distinto de la unidad. Para ello, bastará con añadir la sentencia **STEP seguida de la expresión numérica 3** a la sentencia FOR ... TO, donde la expresión numérica 3 determinará el salto que ha de dar la variable al llegar a NEXT, quedando la estructura de la sentencia así:

FOR variable = {expres. numér.1} TO {expres. numér.2} STEP {expres. numér.3} /NEXT variable

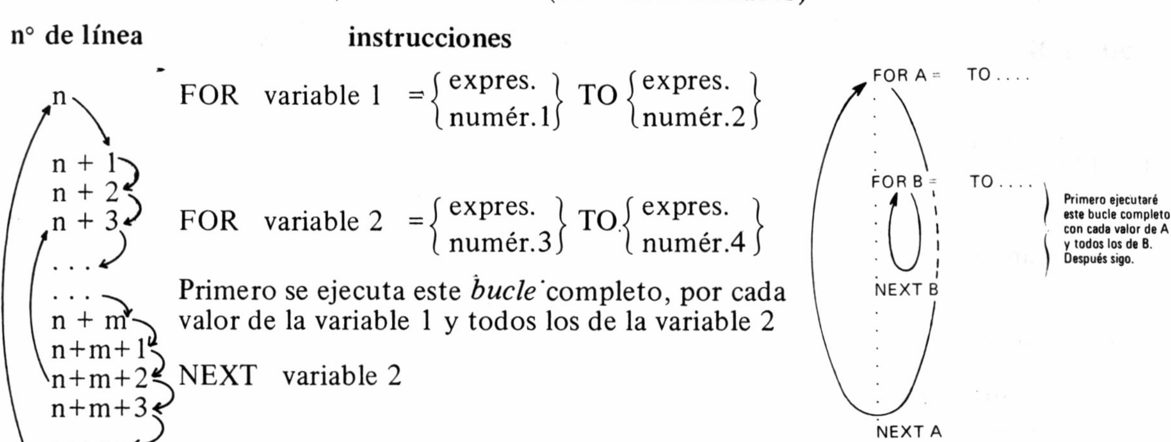
En esta secuencia, la variable que sigue a FOR sólo puede ser una letra.

Esta sentencia produce un *bucle de lectura* que se repite, una y otra vez, hasta que la variable alcanza el valor de la expresión numérica 2, pudiéndose representar esquemáticamente de la siguiente forma:

Bucle FOR ... TO/NEXT normal (de unidad en unidad)

n° de línea	**instrucciones**
n	FOR variable = {expres. numér.1} TO {expres. numér.2}
n + 1	
n + 2	
. . .	
n + n	NEXT variable

Bucle FOR ... TO ... STEP/NEXT unidades (de m en m unidades)



La secuencia normal de lectura se restablece cuando la variable 1 se hace igual a la expresión numérica 2. En ese momento, la siguiente línea a leer es aquélla que sigue a la sentencia NEXT.

FORMA de teclear la instrucción:	**FOR: Apretar F en modo K** **TO: Apretar F y SIMBOL SHIFT en modo K** **STEP: Apretar D y SYMBOL SHIFT en modo K** **NEXT: Apretar N en modo K**

EJEMPLO:

1. Confeccionar un programa que, una vez ejecutado, imprima en pantalla los resultados de la tabla de multiplicar por 2 desde Ø hasta 1Ø.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø FOR Y = Ø TO 1Ø	En este programa nombramos la variable con la letra Y; la expresión numérica 1 la hacemos igual a Ø, y la expresión numérica 2 la hacemos igual a 1Ø.
2Ø LET A = 2 * Y	En la línea 2Ø, damos a la variable A el valor que resulte de multiplicar por 2 el valor actual de la variable Y –que tomará sucesivamente los valores Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9 y 1Ø–.
3Ø PRINT A	**La línea 3Ø** obliga a imprimirse el valor actual de A.

4Ø NEXT Y **5Ø STOP**	**Al llegar a** la línea 4Ø, el programa seguirá su lectura en la línea 1Ø, siempre y cuando el valor de Y no haya llegado a 1Ø. En otro caso, pasaría a la línea 5Ø y el programa parará.

EJEMPLO:

2. Escribir un programa que, una vez ejecutado, imprima en pantalla los números impares del 1 al 5Ø.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø FOR X = 1 TO 5Ø STEP 2 **2Ø PRINT X** **3Ø NEXT X**	En este caso, obligamos a saltar a la variable X de dos en dos –gracias a STEP 2–, partiendo del valor 1 y hasta llegar al valor 5Ø.

GO SUB/RETURN	GO SUB número de línea	IR A SUBRUTINA/VOLVER

INTERPRETACION:

La lectura del programa salta incondicionalmente a un número de línea y continúa a partir de ella hasta encontrar la sentencia

RETURN

con la que se produce un salto incondicional a la línea de programa siguiente a aquélla que contenía la sentencia GO SUB anterior.

INTERPRETACION:

Estas sentencias son utilizadas cuando una parte de un programa se repite varias veces y, consiguientemente, es más cómodo considerar esa parte del programa como una *subrutina* a la que poder dirigirse –gracias a GO SUB– tantas veces como sea necesario.

También se suele utilizar GO SUB para organizar un programa en *subrutinas*, de tal forma, que su estructura sea clara.

FORMA de teclear la instrucción: **GOSUB: Apretar H en modo K**
RETURN: Apretar Y en modo K

EJEMPLO:

1. Para ser candidata a Miss Universo se debe medir más de 1,70 metros y pesar menos de 70 kilogramos.

   ¿Podría hacer un programa en el que dado el nombre, la estatura y el peso de cada aspirante se obtuviera la respuesta de si es o no admitida?

SOLUCION:

PROGRAMA	COMENTARIOS
2 INPUT "¿Alguna candidata? "; O$	En el programa anterior, introducimos las líneas 2, 4. 6 y 8.
4 IF O$ = "SI" THEN GO SUB 1Ø	Si el valor de O$ es SI, ENTONCES el programa se dirige a la subrutina que comienza en la línea
6 IF O$ = "NO" THEN STOP	1Ø. Pero si el valor de O$ es NO, ENTONCES el programa termina
8 GO TO 2	
1Ø INPUT "¿Nombre? "; N$	En el programa anterior, borramos las líneas 7Ø,
2Ø INPUT "¿Talla? "; T	8Ø y 12Ø. Para hacerlo, basta con teclear cada uno
3Ø INPUT "¿Peso? "; P	de estos tres números seguido de ENTER, con lo

PROGRAMA	COMENTARIOS

```
40 IF T < 1.7 THEN GO TO 110
50 IF P > 60 THEN GO TO 110
60 PRINT N$, "admitida"
70 RETURN
90 CLS
100 GO TO 10
110 PRINT N$, "rechazada"
120 RETURN
```

que cada línea desaparecerá automáticamente del listado.
Añadimos las líneas:
70 RETURN y
120 RETURN
También podíamos haber tecleado estas dos líneas directamente, con lo que hubieran sustituido a las que teníamos antes sin necesidad de borrarlas.
Dejamos el resto del programa como estaba.

GO TO	GO TO número de línea	IR A

INTERPRETACION:

Se ordena un salto incondicional a un número de línea.

POSIBILIDADES:

La secuencia de lectura de un programa, por parte del ordenador, sigue un orden creciente, yendo del número de línea más pequeño al mayor, de una forma sucesiva. Pero, cuando a lo largo de esta lectura, se encuentra con la orden GO TO, inmediatamente, el orden normal de la lectura se transfiere a la línea cuyo número está indicado en el número de línea.

FORMA de teclear la instrucción: **Apretar G en modo K**

EJEMPLO:

1. Hagamos utilizable indefinidamente el programa que determinaba el importe de una conferencia.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø INPUT "¿Número de pasos? ";N **2Ø INPUT "¿Valor de un paso? ";P** **3Ø PRINT "El importe de su conferencia es ";N * P** **4Ø GO TO 1Ø**	Al añadir la línea 4Ø, obligamos seguir la lectura del programa a partir de la línea 1Ø tantas veces como queramos.

IF/THEN	**IF expresión THEN sentencia**	**SI ... / ENTONCES ...**

INTERPRETACION:

SI el resultado de la expresión es *cierto* (o *falso*) **ENTONCES** obedece la orden dada por sentencia.

POSIBILIDADES:

La expresión siempre estará formada por operadores de relación u operadores de relación y operadores lógicos.

La sentencia puede ser cualquiera de las que figuran en el ESQUEMA DE COMANDOS BASIC.

FORMA de teclear la instrucción: **IF: Apretar U en modo K**
THEN: Apretar G y SYMBOL SHIFT en modo K

EJEMPLO:

1. Para ser candidata a Miss Universo se debe medir más de 1,70 metros y pesar menos de 60 kilogramos.

   ¿Podría hacer un programa en el que dado el nombre, la estatura y el peso de cada aspirante se obtuviera la respuesta de si es o no admitida?

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø INPUT "¿Nombre? "; N$	El programa se interrumpirá durante su ejecución hasta que se introduzca la cadena de caracteres que represente el nombre.
2Ø INPUT "¿Talla? "; T	Se producirá una nueva interrupción hasta que se introduzca el valor numérico correspondiente a la talla.
3Ø INPUT "¿Peso? "; P	Se producirá otra interrupción hasta que se teclee el valor numérico del peso.
4Ø IF T < 1.7 THEN GO TO 11Ø	SI el valor dado a T es menor que 1.7 ENTONCES, y sólo entonces, el programa salta a la línea 11Ø.
5Ø IF P > 6Ø THEN GO TO 11Ø	SI el valor dado a la variable P es mayor que 6Ø ENTONCES, y sólo entonces, el programa salta a la línea 11Ø.
6Ø PRINT N$, "admitida"	Si el ordenador llega a leer esta instrucción es, evidentemente, porque en la línea 4Ø el valor de T

PROGRAMA	COMENTARIOS
	era mayor que 1.7, y en la línea 5Ø el valor de P era menor que 60, con lo cual la candidata supera las condiciones impuestas de talla y peso. Por tanto, se ordena la impresión de su nombre y el mensaje *admitida.*
7Ø INPUT "¿Otra candidata? "; O$	Con esta interrupción, el programa queda a la espera de una cadena de caracteres, cuya función se ve en la línea siguiente.
8Ø IF O$ = "NO" THEN STOP	SI la cadena asignada a la variable O$ es igual a NO, entonces el programa se detiene, ya que así lo ordena la sentencia STOP.
9Ø CLS	Esta sentencia –que se explicará en otro momento– ordena un borrado de pantalla, con el que desaparecen todas las impresiones previas.
1ØØ GO TO 1Ø	Para que el ordenador llegue a leer esta instrucción habrá sido necesario que el valor dado a O$ haya sido distinto a NO y, de esta forma, habremos mandado la lectura del programa a la línea 1Ø, comenzando de nuevo el proceso.
11Ø PRINT N$, "rechazada"	Esta línea de programa sólo será leída si alguna de las líneas 4Ø ó 5Ø lo permiten.
12Ø GO TO 7Ø	En esta línea se provoca un salto incondicional a la línea 7Ø para saber si el proceso debe repetirse.

INTERPRETACION:

Se obtiene el carácter de la tecla que se aprieta o la cadena vacía, en el momento en que se ejecuta la sentencia INKEY$. Dicho de otro modo, si al llegar la lectura del programa a la función INKEY$ hay alguna tecla apretada, entonces se obtendrá el carácter de dicha letra; en caso contrario, el computador la considerará cadena vacía.

> Se entiende por **cadena vacía** aquélla que no contiene ningún carácter. Se representa por dos comillas consecutivas **(" ")**, de tal modo que si escribimos, por ejemplo, **LET A$** = " ", el computador considerará que la variable de caracteres A$ no contiene ninguno.

POSIBILIDADES:

Esta función se puede aplicar directamente o a través de una variable de caracteres. Esto quiere decir que INKEY$, al ser una función y, por tanto, devolver un valor —un carácter, en este caso—, puede ser manejada como una cadena independiente, haciéndola igual a una variable alfanumérica o ella por sí misma.

FORMA de teclear la instrucción: **Apretar N en modo E**

EJEMPLO:

Diseñar un bucle, dentro de un hipotético programa, que impida seguir la lectura del mismo hasta que se pulse una tecla cualquiera.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø .... **3Ø IF INKEY$** = " " **THEN GO TO 3Ø** **5Ø**	En función de lo expuesto anteriormente, en la línea 3Ø imponemos la condición de que mientras INKEY$ dé como resultado la cadena vacía, la lectura del programa vaya a su propia línea. Es decir, mientras no se apriete una tecla cualquiera, el programa quedará detenido en la línea donde está INKEY$.

EJEMPLO:

Escribir un programa que, al ejecutarlo, pregunte: "¿Es Londres la capital de Inglaterra?" y, a continuación, "Apriete S en caso afirmativo".

Desarrolle el listado para emitir mensajes de acuerdo con la corrección o incorrección de la respuesta.

SOLUCION:

PROGRAMA	COMENTARIOS
1∅ PRINT "¿Es Londres la capital de Inglaterra?" **2∅ PRINT** **3∅ PRINT "Apriete S en caso afirmativo."** **4∅ LET R$ = INKEY$: IF R$ = " " THEN GO TO 4∅** **5∅ IF R$ = "S" THEN PRINT "Correcto": STOP** **6∅ PRINT "Incorrecto"** **7∅ STOP**	En la línea 4∅, la ejecución del programa queda en un bucle sobre sí mismo, según se vió en el ejemplo anterior, pero utilizando INKEY$ indirectamente, gracias a R$. Tanto en la línea 4∅ como en la 5∅, R$ sólo puede contener un carácter –el de la tecla apretada, dado por INKEY$– o ninguno –cadena vacía–.

INPUT	INPUT "texto" ; variable	ENTRADA

INTERPRETACION

Produce una interrupción en la ejecución de un programa permitiendo, de este modo, la entrada de información a través del teclado.

POSIBILIDADES:

Cuando un programador introduce un comando INPUT, está provocando una parada en el programa para, de esta forma, dar un contenido a la variable.

Si el "texto" ha sido escrito, éste aparecerá en la pantalla durante la interrupción y hasta que el dato solicitado haya sido tecleado y se apriete la tecla ENTER.

Si el "texto" no existe, sólo el *prompt* (1) típico del ordenador aparecerá en pantalla, indicando que está a la espera de información. En este caso, el dato que se introduzca será asignado a la variable.

Esta variable puede ser numérica o de caracteres y, consiguientemente, el dato introducido tiene que estar en consonancia con el tipo de variable. Es decir, si la variable se ha considerado como numérica, no se aceptará una cadena de caracteres como entrada del INPUT.

Si a continuación de INPUT colocamos LINE (apretar 3 y SYMBOL SHIFT en modo E) seguido de una variable de caracteres, el ordenador quedará a la espera de una cadena de caracteres, pero en pantalla – al *correr* el programa– no aparecerán las comillas de la instrucción equivalente que sería: INPUT seguido de la variable de caracteres.

FORMA de teclear la instrucción: **Apretar I en modo K**

EJEMPLO:

1. Escriba un programa que, una vez ejecutado, nos pida el número de "pasos" consumidos en una conferencia telefónica y, a continuación, el valor de cada "paso", para obtener finalmente el importe de la conferencia.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø INPUT "¿Número de pasos? " ; N	Con las dos primeras líneas provocaremos, al ejecutar el programa, dos interrupciones para asignar valores a las variables **N** y **P.**

(1) Con la expresión *prompt* se viene a indicar que el ordenador está preparado para hacer su trabajo y, para ello, hace aparecer en pantalla un símbolo, al que por extensión se llama así.

PROGRAMA	COMENTARIOS
2Ø INPUT "Valor de un paso? "; P	
3Ø PRINT "El importe de su conferencia es ";	Con la tercera línea, ordenamos la impresión del texto que figura tras el comando PRINT.
4Ø PRINT N * P	Con la cuarta línea, ordenamos la impresión del resultado de multiplicar el contenido actual de las variables N y P.

Cuando ejecutemos este programa, tecleando RUN y ENTER, en la pantalla aparecerá escrito **¿Número de pasos?** y se parará a la espera del contenido de la variable **N.**

Si suponemos que el número de pasos ha sido **25**, tecleamos este número y, a continuación, ENTER, para indicar al ordenador que hemos concluido, aparecerá escrito en la pantalla **¿Valor de un paso?** y volverá a pararse a la espera del contenido de la variable **P**.

Si asignamos a esta variable el valor **1Ø, lo cual implica teclear 1Ø** seguido de ENTER, aparecerá en la pantalla **El importe de su conferencia es 25Ø.**

Observe que el resultado de **N * P** se imprime justo a continuación del texto de la línea 3Ø, debido a que éste concluye en el programa con un punto y coma (;).

EJEMPLO:

2. Escriba un programa que, una vez ejecutado, le pida el nombre de un escritor famoso; a continuación, le pida el título de una de sus obras, y, finalmente, imprima en pantalla ambas cosas.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø INPUT "Nombre del escritor? ";e$** **2Ø INPUT "Título de una obra suya? "; t$ **3Ø PRINT e$** **4Ø PRINT t$**	En las líneas 1Ø y 2Ø hemos definido el nombre de dos variables de caracteres. En las líneas 3Ø y 4Ø hemos ordenado la impresión de las variables de caracteres definidas anteriormente, una debajo de la otra. Sustituya la línea 1Ø por esta otra y compare: **1Ø INPUT "Nombre del escritor?"; LINE e$**

INT (expresión numérica)

INTERPRETACION:

Se obtiene el mayor de los números enteros menores —o igual— que el indicado en la expresión numérica.

POSIBILIDADES:

El valor de la expresión numérica puede ser cualquier número real.

FORMA de teclear la instrucción: **Apretar R en modo E**

EJEMPLOS:

PROGRAMA 1

```
10 LET x = 89.98
20 PRINT INT x
```

COMENTARIOS

Al ejecutar este programa, obtendremos 89 impreso en la pantalla, que es el mayor número entero menor que 89.98.

PROGRAMA 2

```
10 LET x = -10.17
20 PRINT INT x
```

COMENTARIOS

Al ejecutar este programa, obtendremos –11 impreso en la pantalla, que es el mayor número menor que –10.17.

PROGRAMA 3

```
10 LET x = 10.17
20 PRINT INT x
```

COMENTARIOS

En este caso, obtendremos 10 impreso en la pantalla, que es el mayor número entero menor que 10.17.

LEN	LEN (expresión alfanumérica)	LONGITUD

INTERPRETACION:

Se obtiene el número de caracteres que componen la expresión alfanumérica, incluyendo los espacios.

POSIBILIDADES:

El resultado que se obtiene con LEN es, a todos los efectos, un número y, por tanto, podemos utilizarlo como tal según convenga.

También podemos usar LEN en la suma de cadenas o de variables de caracteres.

FORMA de teclear la instrucción: **Apretar K en modo E**

EJEMPLO:

Sumar 1Ø al número de caracteres que contenga la suma de la variable **a$** y la cadena "**ABC**", suponiendo que **a$** = "**ZXY**".

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø LET a$ = "ZXY"	En la primera línea, asignamos a la variable de caracteres **a$** la cadena indicada "**ZXY**".
2Ø LET a = LEN (a$ + "ABC")	En la línea 2Ø, asignamos a la variable numérica **a** el número correspondiente al total de caracteres existente entre la cadena representada por a$ y la "**ABC**".
3Ø PRINT a + 1Ø	En la última línea, ordenamos la impresión del resultado de la suma de la variable **a** más 1Ø.

LET	LET variable = expresión	DEJAR

INTERPRETACION

Asigna a una variable el valor de una expresión.

POSIBILIDADES:

Recordemos previamente que una *variable* contiene un *valor* que puede cambiar a lo largo de un algoritmo, pero que permanece fijo mientras no haya una instrucción que así lo determine.

Existen dos clases de variables: *numéricas* y *de caracteres.*

Las variables numéricas se representan por cualquier conjunto de caracteres –a veces, por una sola letra seguida de un número– con la condición de que el primero sea una letra. Ejemplo: LET ABc1 = *45.*

Las variables de caracteres se representan por una sola letra seguida del símbolo $. De esta forma, el computador sabe que el contenido de la variable será un texto, que deberá ser introducido entre comillas. Ejemplo: LET A$ = "Cervantes".

Al referirnos a las variables, dijimos que son elementos capaces de contener un valor. *Una expresión,* por lo contrario, *es un valor.*

Nada impide que el valor de una expresión varíe de acuerdo con los valores de las variables que forman parte de la expresión.

FORMA de teclear la instrucción: **Apretar L en modo K**

EJEMPLO:

1. Definir una expresión que determine el valor del importe de una conferencia con Sevilla, sabiendo que el valor de cada "paso" de contador lo podemos averiguar en la Telefónica.

SOLUCION:

Empezaremos por fijar y nombrar las variables que intervienen:

Llamaremos **P** a la variable que contendrá el valor de un "paso".

Llamaremos **N** a la variable que contendrá el número de "pasos".

Y llamaremos **I** a la variable que contendrá el importe de la conferencia con Sevilla.

Hecho esto, tendremos que definir *la expresión* que determine el valor del importe, en función de los valores que contengan las variables implicadas:

I = P * N

Es evidente que el importe de la conferencia será igual al resultado de multiplicar el número de "pasos" por el valor de un "paso".

Supongamos que, después de observar en el contador el número de pasos y de habernos informado en la Telefónica del valor de un "paso", los valores de las variables **P** y **N** son, respectivamente, **10** y **25**. Es decir, un "paso" vale 10 ptas. y el número de "pasos" de la conferencia ha sido 25. En función de esto, la forma de asignar variables en BASIC, teniendo en cuenta el comando LET, será:

```
LET P = 10
LET N = 25
LET I = P * N
```

Como podemos ver, la expresión en una variable numérica puede ser tanto un número como una expresión matemática, la cual, finalmente, es un número.

EJEMPLO:

2. Asignar a una variable de caracteres la expresión:
   El importe de su conferencia es.

SOLUCION:

LET A$ = "El importe de su conferencia es "

De esta forma, el ordenador sabe que la variable A$ contiene el texto entrecomillado.

Observe el espacio que queda entre *es* y las comillas. Sirve para evitar que la cifra que representa el valor de la conferencia quede impresa junto a *es*.

EJEMPLO:

3. Escriba un programa que, una vez ejecutado, nos dé el importe de una conferencia con Sevilla de 25 "pasos", siendo el valor de cada paso 10 ptas.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø LET P = 1Ø **2Ø LET N = 25** **3Ø LET I = P * N** **4Ø LET A$ = "El importe de su conferencia es "** **5Ø PRINT A$; I**	Con las cuatro primeras líneas asignamos nombre y contenido a las variables. Con la última línea nos limitamos a ordenar la impresión del contenido actual de las variables A$ e I.

Al ejecutar (RUN/ENTER) el programa anterior, aparecerá en pantalla:

El importe de su conferencia es 250

LIST	**LIST número de línea**	**LISTAR EN PANTALLA**
LLIST	**LLIST número de línea**	**LISTAR EN IMPRESORA**

INTERPRETACION:

LIST ordena listar en pantalla todas las líneas del programa a partir del número de línea.

Con LLIST se consigue el mismo resultado pero impreso en papel, mediante una impresora conectada al computador.

POSIBILIDADES:

Si el número de línea no se da, se obtiene un listado del programa desde el principio hasta el final.

Teclear **LIST o LLIST, seguido de ENTER,** para obtener *el listado completo del programa* en pantalla o en papel.

Teclear **LIST o LLIST y el número de línea,** seguido de *ENTER, para obtener el listado del programa desde la línea del número dado,* en pantalla o en papel.

FORMA de teclear la instrucción: **LIST: Apretar K en modo K**
LLIST: Apretar V en modo E

LN	LN (expresión numérica)	LOGARITMO

INTERPRETACION:

Se obtiene el valor del logaritmo natural de la expresión numérica.

POSIBILIDADES:

El valor de la expresión numérica debe ser mayor que Ø.

FORMA de teclear la instrucción: **Apretar Z en modo E**

EJEMPLO:

Obtenga el logaritmo en base **e** de 6.5, utilizando su ordenador en forma directa.

SOLUCION:

PRINT LN 6.5

Pulse ENTER y obtendrá la respuesta.

LOAD	LOAD "nombre"	CARGAR

INTERPRETACION:

Instruye al computador para que cargue en su memoria RAM información procedente de un soporte de memoria externo del tipo casete o microcinta.

POSIBILIDADES:

LOAD " " carga el primer programa que encuentra en la cinta.

LOAD *"nombre"* carga exclusivamente aquel programa que haya sido grabado (salvado) en cinta con el nombre *"nombre"*.

LOAD *"nombre"* **DATA** *denominación de la matriz* ( ). La *denominación de la matriz* se ajusta a lo indicado en la sentencia **DIM**, de forma que las matrices numéricas requerirán una letra para su denominación y las alfanuméricas una letra seguida del símbolo $. Con esta instrucción se carga la matriz que haya sido grabada (salvada) en cinta con el nombre *"nombre"* y la reconocerá por la *denominación de la matriz.*

LOAD " " CODE carga el primer programa en código máquina que encuentra en la cinta.

LOAD *"nombre"* **CODE** carga el programa en código máquina grabado (salvado) en cinta con el nombre *"nombre"*

LOAD *"nombre"* **CODE** *dirección de comienzo, longitud* cumple la misma función que la instrucción anterior, por tanto, esta instrucción se utiliza cuando la dirección de memoria desde donde queremos iniciar la carga del programa *"nombre"* en código máquina es distinta de aquella en la que fue grabada (salvada).

LOAD *"nombre"* **SCREEN$** carga directamente en el fichero de imágenes la *pantalla* que haya sido grabada (salvada) en cinta con el nombre *"nombre"*. Si *"nombre"* no existe y sólo las comillas son utilizadas, carga la primera *pantalla* que encuentre en la cinta.

LOAD * "m"; *nº microdrive;* *"nombre"* nos servirá para la carga procedente del *microdrive,* teniendo en cuenta que *nº microdrive* debe indicar el microdrive que debe ser manipulado por el ordenador y que *todas* las referencias generales indicadas en la instrucción **deben** figurar, por lo demás es válido lo dicho anteriormente refiriéndonos al casete.

FORMA de teclear la instrucción: **Apretar I en modo K**

MERGE	MERGE "nombre"	FUNDIR (Mezclar)

INTERPRETACION:

Mezcla un programa en BASIC procedente de cinta (o microcinta) con el que actualmente esté en memoria, sustituyendo en éste aquellas líneas de programa que –procedentes del programa a fundir – coincidan.

POSIBILIDADES:

MERGE"" Mezcla con el programa actualmente en memoria el primero que encuentre en la cinta.

MERGE *"nombre"* cumple la misma función, pero sólo con un programa que previamente haya sido salvado con el nombre *"nombre"*.

MERGE * "m"; *nº micridrive; "nombre"* donde *nº de microdrive* indica el microdrive donde está instalada la microcinta que contiene el programa a mezclar. Todas las referencias indicadas deben aparecer.

FORMA de teclear la instrucción: **Apretar T y SYMBOL SHIFT en modo E**

NEW — NUEVO

INTERPRETACION:

Borra el programa que esté en la memoria RAM del computador y todas las variables.

POSIBILIDADES:

Cuando se ejecuta la instrucción LOAD, el resultado –con respecto a un programa que puediera haber en la memoria RAM– es el mismo que si se ejecuta previamente un NEW.

FORMA de teclear la instrucción: **Apretar A en modo K**

PRINT	PRINT expresiones	IMPRIMIR

INTERPRETACION:

Significa *"imprimir"* y ordena la impresión en pantalla de la expresión o expresiones que figuran, optativamente, en las expresiones.

POSIBILIDADES:

* Si las expresiones son omitidas, una línea de pantalla salta en blanco.

** Si las expresiones son incluidas y estas expresiones son cadenas de caracteres, debemos escribirlas entrecomilladas.

*** Si las expresiones son incluidas y estas expresiones son matemáticas, aparecerán en pantalla sus valores. Recuerde que las expresiones matemáticas no se escriben entre comillas.

La posición de impresión de cada expresión de expresiones viene determinada por los símbolos que separan una expresión de otra:

– Un punto y coma (;) hace que la siguiente expresión se imprima justo a continuación de la inmediata anterior.

– Una coma (,) obliga a imprimir a partir de la siguiente mitad de la pantalla o al comienzo de la línea siguiente si la otra mitad ya se ocupó.

– Un apóstrofe imprime la expresión que la sigue al comienzo de la siguiente línea.

Si las expresiones finalizan con una (,) o un punto y coma (;), la siguiente instrucción PRINT, que pudiera aparecer a lo largo del programa, comenzaría la impresión de sus expresiones conforme al criterio expuesto más arriba.

En caso de no acabar con una coma o un punto y coma, la siguiente instrucción PRINT comenzaría la impresión de sus expresiones en la siguiente línea de pantalla que esté libre.

FORMA de teclear la instrucción: **Apretar P en modo K**

EJEMPLO:

1. Escriba un programa que imprima en pantalla la cadena de caracteres: **"La loba lamía, lentamente, los lobeznos 3 veces'."**

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø PRINT "La loba ";	Por supuesto también bastaría con escribir PRINT "La loba lamía, lentamente, los lobeznos 3 veces"

COMENTARIOS

- Teclear **1Ø**
- Teclear **PRINT**
- Teclear “ (comillas)
- Escribir el texto
- Dar un espacio
- Teclear ” (comillas)
- Teclear ; (punto y coma)
- Apretar **ENTER**

> Una vez seguidas las instrucciones dadas en COMENTARIOS, la línea número 1Ø de nuestro primer programa estará en la memoria del ordenador. En este caso, tenemos un programa, compuesto por una sola línea, cuya misión es imprimir el texto indicado cada vez que se le ordene al computador.

De una forma similar iríamos tecleando las siguientes líneas del programa:

PROGRAMA	COMENTARIOS
2Ø PRINT “lamía, lentamente, ”;	• Teclear **2Ø** • Teclear **PRINT** • Teclear “ (comillas) • Escribir el texto • Dar un espacio • Teclear ” (comillas) • Teclear ; (punto y coma) • Apretar la tecla **ENTER**
3Ø PRINT “los lobeznos ”;	• Teclear **3Ø** • Teclear **PRINT** • Teclear ” • Escribir el texto • Dar un espacio • Teclear ” • Teclear ; • Apretar **ENTER**
4Ø PRINT “3 veces.”	• Teclear **4Ø** • Teclear **PRINT** • Teclear • Escribir el texto • Teclear ” • Apretar **ENTER**

RANDOMIZE	RANDOMIZE **expresión numérica**	ALEATORIZAR

INTERPRETACION:

Inicializa el generador de números aleatorios de acuerdo con la expresión numérica, con lo que la secuencia de números RND comenzará con valores distintos.

POSIBILIDADES:

El valor de la expresión numérica debe estar comprendido entre Ø y 65535, con lo que las secuencias RND comenzarán en diferentes lugares.

RANDOMIZE Ø obliga a RND a producir números lo más aleatorios posible.

En algunos desarrollos técnicos y/o científicos, convendrá simular fenómenos aleatorios controlados, de forma tal que, variando algunos parámetros, se repita el proceso con los mismos valores.

FORMA de teclear la instrucción: **Apretar T en modo K**

EJEMPLO:

1. Escriba un programa que produzca números lo más aleatorios posible, independientemente de que se ejecute o se inicialice.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø RANDOMIZE Ø **2Ø FOR X = 1 TO 2Ø** **3Ø PRINT RND** **4Ø NEXT X**	Córralo varias veces y comprobará que, a diferencia del segundo ejemplo expuesto en la ficha de RND, los resultados son diferentes en cada ocasión.

EJEMPLO:

2. Basándose en una misma secuencia de números aleatorios, obtenga las series de resultados que se deriven de los diferentes valores que se den por teclado.

SOLUCION:

PROGRAMA	COMENTARIOS
1Ø INPUT A	Entrada de valores.
15 RANDOMIZE 1	Se fija el punto de partida del generador de números aleatorios.
2Ø FOR X = 1 TO 2Ø **3Ø PRINT RND * A** **4Ø NEXT X** **5Ø GO TO 1Ø**	El resto del programa está dedicado al proceso en estudio que, en este caso, es una simple multiplicación del valor dado en la línea 1Ø por el RND correspondiente.

REM	REM aclaraciones	REMEMORAR

INTERPRETACION:

Significa *rememorar, recordar* y permite introducir aclaraciones y observaciones que pueden ser de interés en una posterior revisión o lectura del listado del programa.

La sentencia REM no implica ningún tipo de proceso.

POSIBILIDADES:

Como se desprende de la nomenclatura utilizada –aclaraciones–, el texto que se utilice para las aclaraciones y observaciones es opcional. Esto quiere decir que si tecleamos REM en una línea de programa –bien a su comienzo, bien tras el separador (:)–, ésta quedará impresa para servir de indicador nemotécnico. En este sentido, es frecuente utilizar el separador (:) como medio de fragmentar un listado en "trozos" físicos para distinguir de un vistazo los diferentes elementos de un programa.

En todo caso, la sentencia REM se mantiene en el listado, tal como se ha escrito, sin ser procesada, continuando la ejecución del programa en la primera sentencia que la siga, bien tras un separador (:), bien en la siguiente línea del programa.

FORMA de teclear la instrucción: **Apretar E en modo K**

EJEMPLO:

Diferentes formas de introducir aclaraciones y observaciones en un listado:

PROGRAMA	COMENTARIO
1Ø REM Comienzo del programa.	REM con aclaraciones.
2Ø INPUT A	
3Ø REM	REM en solitario.
4Ø INPUT B	
5Ø :	Separadores para fragmentar el listado.
6Ø REM Suma	REM con aclaraciones
7Ø LET C = A + B: REM contador	REM tras un separador.
8Ø REM Fin: STOP	REM antes de un separador.
9Ø :	Separadores en forma de línea.

RESTORE	RESTORE número de línea	VOLVER A ALMACENAR

INTERPRETACION:

Con esta sentencia se permite comenzar la lectura de las constantes desde la primera del DATA situado en un número de línea.

Si el número de línea no se especifica, la lectura de constantes se iniciará en la primera del primer DATA del programa.

POSIBILIDADES:

Después de que un programa lee una sentencia RESTORE, la primera sentencia READ que ejecute el programa accederá a la primera constante, según el criterio expuesto en INTERPRETACION.

Teclear **RESTORE y el número de línea**, seguido de **ENTER,** para volver a leer las constantes que hay en el *DATA de la línea del número dado.*

Teclear **RESTORE**, seguido de **ENTER**, para volver a leer las constantes desde *el primer DATA del programa.*

FORMA de teclear la instrucción: **Apretar S en modo E**

EJEMPLOS:

PROGRAMA

```
1Ø FOR X = 1 TO 5
2Ø READ V: PRINT V
3Ø RESTORE
4Ø NEXT X
1ØØ DATA 3
```

COMENTARIOS

La variable V es leída cinco veces en la línea 2Ø aunque tiene una sola constante en el DATA de la línea 1ØØ. Esto es posible gracias al RESTORE de la línea 3Ø, que obliga a comenzar la lectura en el primer valor del primer DATA.

PROGRAMA

```
1Ø FOR X = 1 TO 1Ø
2Ø READ V$ : PRINT V$
3Ø RESTORE
4Ø NEXT X
1ØØ DATA "caballo", "jinete"
```

COMENTARIOS

La variable V$ es leída diez veces en la línea 2Ø tomando el primer valor del DATA de la línea 1ØØ.

PROGRAMA

```
1Ø FOR X = 1 TO 5
2Ø READ V$ : PRINT V$
```

COMENTARIOS

Imprime cinco veces *montar,* primera (en este caso, única) constante del DATA situado en la

PROGRAMA	COMENTARIOS
3Ø RESTORE 9Ø **4Ø NEXT X** **9Ø DATA "montar"** **1ØØ DATA "jinete", "caballo"**	línea 9Ø. Si fuera RESTORE 1ØØ, la impresión en pantalla sería: *montar* *jinete* *jinete* *jinete* *jinete*

RND/RANDOMIZE		ALEATORIO/ALEATORIZAR

RND	RND	ALEATORIO

INTERPRETACION:

Cada vez que se utiliza RND, se obtiene un valor diferente y aleatorio igual o mayor que Ø y menor que 1.

POSIBILIDADES:

Esta función da la misma secuencia de valores cada vez que se ejecuta el programa que la contiene, tecleando RUN/ENTER.

FORMA de teclear la instrucción: **Apretar T en modo E**

EJEMPLOS:

1. Teclee en forma directa estas tres instrucciones y observe los resultados:

   PRINT RND y pulse ENTER
   PRINT RND y pulse ENTER
   PRINT RND y pulse ENTER

2. Teclee este programa en su computador y córralo varias veces, comparando los resultados:

```
1Ø FOR X = Ø TO 2Ø
2Ø PRINT RND
3Ø NEXT X
```

3. Teclee en forma directa estas instrucciones y observe los resultados:

   PRINT RND * 1Ø y pulse **ENTER**
   PRINT RND * (5 + 2) y pulse **ENTER**
   PRINT (5 + 2) * RND y pulse **ENTER**
   PRINT (1.3 + Ø.7) * RND y pulse **ENTER**
   PRINT RND * (1.3 + Ø.7) y pulse **ENTER**
   PRINT (–1Ø) * RND y pulse **ENTER**

SAVE	SAVE "nombre"	SALVAR

INTERPRETACION:

Transfiere la información contenida en la memoria RAM de un computador a un soporte de memoria externo, **salvándola** o **guardándola** para ser utilizada posteriormente, transfiriéndola nuevamente al computador mediante LOAD.

Es necesario recordar que la memoria RAM no es permanente y, al desconectar el computador de la corriente, toda la información que contiene se pierde. Gracias a SAVE y a la función que cumple, la información queda salvada, bien sobre cinta magnética, bien sobre disco.

POSIBILIDADES:

SAVE *"nombre"* graba (salva) en cinta (casete) el programa que actualmente esté en la memoria RAM con el nombre *"nombre".*

SAVE *"nombre"* **DATA** *denominación de la matriz* **( )**. La *denominación de la matriz* se ajusta a lo indicado en la sentencia **DIM**, de forma que las matrices numéricas requerirán una letra para su denominación y las alfanuméricas una letra seguida del símbolo $. Con esta instrucción se graba o salva –con el nombre *"nombre"*– la matriz que actualmente reconozca el computador con la denominación *denominación de la matriz.*

SAVE *"nombre"* **LINE** *nº de línea* cumple la misma función que la anterior, pero con la diferencia de que al ser cargado en el computador con un LOAD se autoejecuta a partir de la línea indicada en *nº de línea.*

SAVE *"nombre"* **SCREEN$** graba o salva en cinta todo el contenido que actualmente esté en pantalla.

SAVE *"nombre"* **CODE** *dirección de comienzo, longitud* graba o salva en cinta –con el nombre *"nombre"* – el programa en código máquina –o bytes simplemente– que actualmente está en memoria a partir de la dirección dada por *dirección de comienzo* y con una longitud –medida en bytes– dada por *longitud.*

SAVE * "m"; *nº microdrive; "nombre"* graba o salva en microcinta –con las mismas especificaciones dadas para el casete–, teniendo en cuenta que *nº de microdrive* debe indicar el microdrive dónde está instalada la microcinta donde se ha de producir la grabación y que todas las referencias generales indicadas más arriba deben figurar.

FORMA de teclear la instrucción: **Apretar S en modo K**

Teclear **SAVE** o **SAVE y el nombre del programa entre comillas,** seguido de **ENTER.**

SGN	SGN (expresión numérica)	SIGNO

INTERPRETACION:

Esta función sólo nos devolverá 1, Ø ó –1, según el valor de la expresión numérica sea mayor, igual o menor que Ø, respectivamente.

POSIBILIDADES:

Si la expresión numérica es mayor que Ø, SGN (expresión numérica) nos dará el valor 1.

Si la expresión numérica es igual que Ø, SGN (expresión numérica) nos dará el valor Ø.

Si la expresión numérica es menor que Ø, SGN (expresión numérica) nos dará el valor –1.

FORMA de teclear la instrucción: **Apretar F en modo E**

EJEMPLO:

1. Teclee en forma directa:

PRINT SGN (4–206) y pulse ENTER
Obtendrá como respuesta del computador: –1.

EJEMPLO:

2. Interprete estas líneas:

```
1Ø INPUT A
2Ø INPUT B
3Ø IF SGN (A – B) = 1 THEN GO TO 1ØØØ
4Ø IF SGN (A – B) = – 1 THEN GO TO 2ØØØ
5Ø PRINT "Valores iguales, repita, por favor" : GO TO
. . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
```

SOLUCION:

En las líneas 1Ø y 2Ø se introducen por teclado dos valores.
En la línea 3Ø, saltará a la línea 1ØØØ si SGN (A–B) es 1.
En la línea 4Ø, saltará a la línes 2ØØØ si SGN (A–B) es –1.
En otro caso –SGN (A–B) x Ø–, se emite un mensaje –"Valores iguales, repita, por favor"– y se pasa a la línea 1Ø.

SIN	SIN (expresión numérica)	SENO

INTERPRETACION:

Se obtiene el valor del seno de la expresión numérica, dado el ángulo en radianes.

POSIBILIDADES:

Las que se derivan del cálculo trigonométrico.

FORMA de teclear la instrucción: **Apretar Q en modo E**

EJEMPLO:

Escribir un programa que permita calcular el seno de un ángulo dado en grados sexagesimales.

SOLUCION:

360° < > 2 π. Esto quiere decir que si **n** es el ángulo en grados sexagesimales, los radianes equivalentes son:

$$n \frac{2\pi}{360}$$

PROGRAMA	COMENTARIOS
1∅ INPUT "Grados sexagesimales"; n **2∅ LET X = n * 2 * PI/360** **3∅ PRINT n; " ("; X; ")", SIN X**	Una vez realizada la transformación a radianes (línea 2∅), se imprime el valor del ángulo en grados sexagesimales; a continuación, entre paréntesis, su equivalente en radianes, y, después, el valor del seno.

SQR	SQR (expresión numérica)	RAIZ CUADRADA

INTERPRETACION:

Se obtiene el valor de la raíz cuadrada de la expresión numérica.

POSIBILIDADES:

El valor de la expresión numérica debe ser siempre igual o mayor que ∅. Por razones matemáticas, hay que tener en cuenta que **SQR** (expresión numérica) = (expresión numérica) elevada a (1/2).

FORMA de teclear la instrucción: **Apretar H en modo E**

EJEMPLO:

1. Teclee en forma directa:

   PRINT SQR 25 y pulse ENTER

   Obtendrá como respuesta del computador 5, que es el valor de la raíz cuadrada de 25:

   $\sqrt{25} = 25 \uparrow (1/2) = 5$

EJEMPLO:

2. Determine, mediante un programa, las raíces cuadradas de los números enteros comprendidos entre ∅ y 2∅.

SOLUCION:

PROGRAMA	COMENTARIOS
1∅ FOR X = ∅ TO 2∅ **2∅ PRINT "Raíz cuadrada de "; X; "="; SQR X** **3∅ NEXT X**	Corra este programa tecleando RUN/ENTER y observe la impresión en pantalla.

INTERPRETACION:

Cuando la lectura del programa llega a la instrucción **STOP**, finaliza la ejecución del mismo, pasando el computador a la situación de disponible.

Si el programa tiene más instrucciones después de STOP, para proseguir su ejecución es necesario teclear **CONT** seguido de ENTER.

POSIBILIDADES:

Cuando un programa detiene su ejecución, como consecuencia de un STOP, siempre existe la posibilidad de continuarla si se teclea –como se ha dicho más arriba– el comando CONT seguido de ENTER.

FORMA de teclear la instrucción: **STOP: Apretar A y SYMBOL SHIFT en modo K**
CONT: Apretar C en modo K

EJEMPLO:

PROGRAMA

```
1Ø INPUT A
2Ø PRINT A
3Ø INPUT B
4Ø PRINT B
5Ø PRINT A + B
6Ø STOP
7Ø PRINT (A + B) * 2
8Ø STOP
```

COMENTARIOS

Una vez introducidos por el teclado los dos valores (números) que se solicitan en las variables de las líneas 1Ø y 3Ø, el programa detendrá su ejecución en la línea 6Ø imprimiendo el resultado de la suma.
Si tecleamos CONT/ENTER, seguirá la ejecución del programa imprimiendo el resultado de la operación de la línea 5Ø y se detendrá en la línea 8Ø, donde hay otro STOP.

INTERPRETACION:

Se obtiene una cadena de caracteres compuesta por los mismos dígitos que componen la expresión numérica y en el mismo orden.

POSIBILIDADES:

Por ser el resultado obtenido una cadena de caracteres, no podrá ser manejado como el número que era en la expresión numérica, pero sí —a todos los efectos— como la cadena que es.

FORMA de teclear la instrucción: **Apretar Y en modo E**

EJEMPLO:

PROGRAMA

```
1Ø LET a = 1984
2Ø LET b$ = STR$ a
3Ø PRINT a, b$
```

COMENTARIOS

En la línea 2Ø, asignamos a la variable de caracteres **b$** la cadena equivalente al número contenido en la variable **a**.
Cuando ejecute este programa, observará que, en la misma línea y separadas de acuerdo con la coma existente entre **a** y **b$**, aparece dos veces 1984. Bien, pues la primera es a todos los efectos un número, ya que procede del valor contenido en la variable numérica **a**, mientras que la segunda es una cadena de caracteres.

VAL	VAL (expresión alfanumérica)	VALOR

INTERPRETACION:

Se obtiene el valor numérico de la expresión alfanumérica indicada.

POSIBILIDADES:

Es necesario advertir que para poder aplicar la función VAL –como una transformación de caracteres a dígitos– sobre una expresión alfanumérica ésta debe estar compuesta exclusivamente por dígitos (números del Ø al 9), sin ningún signo entre ellos, ya que si los hubiere efectuaría las correspondientes operaciones.

FORMA de teclear la instrucción: **Apretar J en modo E**

EJEMPLO:

PROGRAMA	COMENTARIOS
1Ø LET a$ = "1984" **2Ø PRINT VAL (a$) + 2**	La respuesta a este programa es 1986, suma del valor de la variable a$ –transformada– más 2.

EJEMPLO:

PROGRAMA	COMENTARIOS
PRINT VAL "1985–1"	Teclee en forma directa y observe el resultado.

VERIFY	VERIFY "nombre"	VERIFICAR

INTERPRETACION:

Comprueba que lo grabado o salvado en la cinta coincide exactamente con el contenido en memoria y que –previamente– ha sido transferido del computador al casete (o microcinta).

POSIBILIDADES:

VERIFY" " compara el programa que actualmente está en memoria con el primer programa que se encuentre en la cinta.

VERIFY *"nombre"* cumple la misma función que la instrucción anterior pero no inicia la comparación hasta que encuentra un programa que haya sido salvado con el nombre *"nombre"*.

VERIFY *"nombre"* **DATA** *denominación de la matriz* compara la matriz denominada *denominación de la matriz* y que actualmente reconoce así el computador con la matriz que haya sido grabada en la cinta con el nombre *"nombre"*.

VERIFY *"nombre"* **CODE** *dirección de comienzo, longitud* compara los *bytes* grabados en cinta previamente con el nombre *"nombre"* a partir de la dirección *dirección de comienzo* y a lo largo de una longitud –medida en bytes– dada por *longitud*.

VERIFY "CODE" cumple la misma función que el anterior, pero comparando con el primer grupo de bytes que, grabados en la cinta, se encuentre.

VERIFY * "m"; *nº microdrive; "nombre"* donde *nº microdrive* indica el microdrive donde está instalada la microcinta. Todas las referencias indicadas deben aparecer, por lo demás es válido lo dicho anteriormente con respecto al casete.

FORMA de teclear la instrucción: **Apretar R y SYMBOL SHIFT en modo E**

Sección III

Programación avanzada

GRAFICOS Y COLOR

Esta Sección del libro está dedicada a ayudar al lector en el camino de transformar sus ideas en programas plenos de color y movimiento.

Aparentemente los programas que se desarrollan en este apartado son simples juegos. No los considere así, ya que, según mi experiencia, los programas que podrían tener el calificativo de "técnicos", suelen ser más fáciles de concebir debido a que, normalmente, están sometidos a fórmulas y procesos muy determinados, en tanto que los programas de "juegos" son trozos de su propia imaginación.

Todo lo expuesto a continuación tiene por misión abrir el campo de posibilidades que usted tiene delante, por tanto le sugiero que cambie y pruebe toda cosa que crea que mejoraría los resultados.

Si consideramos la pantalla como una pizarra donde el Spectrum nos va a dar su respuestas a nuestras cuestiones, vamos a empezar por conocer en que forma la máquina usa esa "pizarra".

Cuando se conecta el computador al televisor, el "cerebro" de aquél divide de forma inmediata, y sin que se pueda apreciar, la pantalla de éste en una malla de 45.086 cuadritos o, lo que es igual, 256 cuadritos horizontales por 176 cuadritos verticales (ver figura 1 en un desplegable entre las páginas 16 y 17). Estos cuadritos pueden estar "encendidos" o "apagados". Y según la combinación de cuadritos encendidos y apagados se verá en la pantalla una figura u otra.

Siguiendo el símil de la pizarra, la instrucción INK equivale al color de la tiza con la que vamos a escribir y la instrucción PAPER nos indica el color de fondo de la pizarra.

Evidentemente, el término **apagado** significa que el cuadrito en cuestión mantiene el color dado por PAPER, mientras que **encendido** significa que toma el color dado por INK.

Para aclarar este concepto tomaremos, por ejemplo, las instrucciones INK 1 y PAPER 2. Debido a la primera los cuadritos encendidos tomarán el color azul, los apagados mantendrán el color general del fondo que, en este caso, es rojo y que es impuesto por la segunda.

Es claro que utilizando colores para INK y PAPER con poco contraste, apenas se podrá distinguir nada. En el caso extremo en el que INK y PAPER tengan el mismo color, no habrá diferencia entre cuadritos encendidos y apagados. En otras palabras escribiríamos con tiza negra sobre una pizarra negra, si hubieramos elegido INK Ø, PAPER Ø.

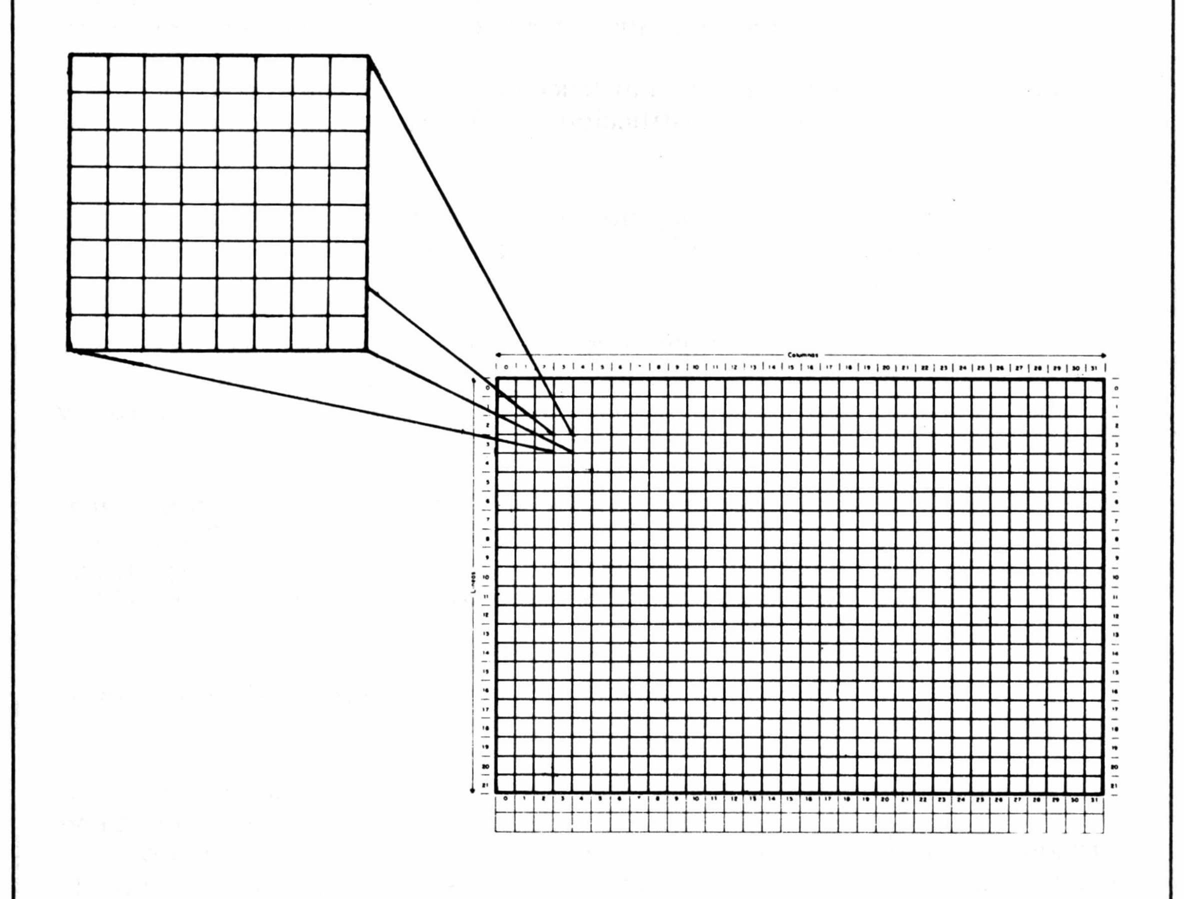
Hasta ahora hemos visto cómo "pinta" el Spectrum sobre la pantalla; veamos ahora cómo escribe.

Anteriormente estudiamos que el Spectrum "contempla" la pantalla como un conjunto de 45.086 cuadritos (256 x 176); pues bien, el Spectrum tiene guardados en su memoria un conjunto de caracteres, que le es propio por diseño, tales como todo el abecedario en mayúsculas y minúsculas, los signos de puntuación, números, etc. y además los tiene definidos en pequeñas mallas de 8 cuadritos horizontales por 8 verticales, de tal forma que, si tuviera que mos-

3-1	COMO GOBIERNA EL SPECTRUM LA PANTALLA DE T.V.

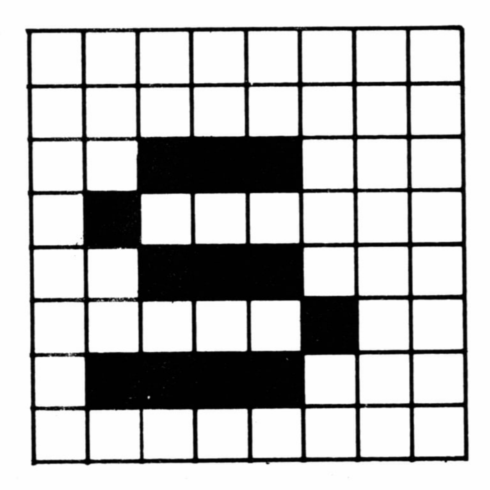
trar en la pantalla uno cualquiera de estos caracteres predefinidos tomaría, dentro de los 45.086 cuadritos totales, una zona de 8 x 8 cuadritos dentro de la cual encendería unos y apagaría otros, mostrando, de esta forma, la figura del carácter deseado.

REPRESENTACION GRAFICA DE LA PANTALLA



De lo dicho hasta aquí se desprende que la pantalla gobernada por el Spectrum nos puede dar 32 caracteres por línea (256/8) y un total de 22 líneas (176/8).

Como ya sabemos, con la instrucción PRINT ordenamos a la máquina que imprima en pantalla números y caracteres en general. De acuerdo con lo visto anteriormente, cuando usamos PRINT para escribir una caracter en la pantalla, lo que estamos haciendo en realidad es decirle al computador que nos muestre cual es el conjunto de cuadritos encendidos y apagados que corresponde a ese caracter, y que él tiene "guardado" en su memoria. Veamos cual es la combinación correspondiente a la **s** por ejemplo:



Estos son los caracteres que están predefinidos en la máquina por el diseñador.

El usuario puede definir sus propios caracteres o modificar los predefinidos. Esto lo veremos más tarde.

De ahora en adelante, cuando hagamos referencia a caracteres en general estaremos hablando del espacio ocupado en pantalla por mallas de 8 x 8 cuadritos. Un espacio en blanco es una malla de 8 x 8 con sus 64 cuadritos apagados.

Recordamos que las filas van aumentando de 0 a 21 de arriba a abajo y las columnas de 0 a 31 de izquierda a derecha.

Para cerrar este capítulo vamos a realizar un ejercicio de impresión en pantalla, interesante desde el punto de vista de la estética y, en muchos casos, de gran utilidad.

3-1	COMO GOBIERNA EL SPECTRUM LA PANTALLA DE T.V.

El siguiente programa nos permite imprimir asteriscos (*) en cualquier punto de la pantalla:

```
  10 INPUT "numero de linea?";l
  20 INPUT "numero de columna?";
c
  30 PRINT AT l,c;"*"
  40 GO TO 10
```

Para imprimir un asterístico en cualquier punto de la mitad izquierda de la pantalla y su simétrico en la mitad derecha, hay que considerar que, si las coordenadas de entrada son **l** y **c**, para su simétrico serán **l** y **31-c:**

```
  10 INPUT "numero de linea?";l
  20 INPUT "numero de columna?";
c
  25 LET c=INT (c/2)
  30 PRINT AT l,c;"*";AT l,31-c;
"*"
  40 GO TO 10
```

Con la línea 25 limitamos **c** a la mitad izquierda de la pantalla.

Para conseguir que el asterisco se imprima según cuatro planos de simetría:

```
  10 INPUT "numero de linea?";l
  20 INPUT "numero de columna?";
c
  25 LET l=INT (l/2): LET c=INT
(c/2)
  30 PRINT AT l,c;"*";AT l,31-c;
"*";AT 21-l,c;"*";AT 21-l,31-c;"
*"
  40 GO TO 10
```

Con la línea 25 limitamos **l** y **c** al cuadrante superior izquierdo.

Supongamos que deseamos ponerle un marco a la pantalla, para ello escribiríamos un programa similar a éste:

```
   10 PRINT "********************
************"
   20 FOR i=1 TO 20
   30 PRINT "*";TAB (31);"*"
   40 NEXT i
   50 PRINT "********************
************"
```

Este marco hecho con asteriscos no queda mal, pero el Spectrum nos provee de 16 caracteres gráficos predefinidos, es decir, están memorizados dentro de la máquina. Ocho de estos caracteres están a la vista en la primera fila de teclas de la máquina y los restantes son, simplemente, sus inversos.

Para obtener los ocho primeros basta con pasar al modo gráfico **(cursor G)** y apretar, a continuación, la tecla correspondiente al caracter deseado. Así, apretaremos simultáneamente las teclas **CAPS SHIFT** y **GRAPHICS** para cambiar el cursor a G y, en estas condiciones, al apretar cualquier tecla con caracter gráfico, nos producirá en la pantalla la impresión de ese carácter.

Para conseguir las inversas tendremos que pasar el cursor igualmente a G y, una vez hecho esto, apretar simultáneamente CAPS SHIFT y la tecla correspondiente.

Compare los gráficos que aparecen en pantalla con los siguientes ejemplos:

Pulse CAPS SHIFT y GRAPHICS. (Ahora el cursor está en G). Pulse la tecla con el número 5, la tecla con el número 6, la tecla con el número 2 y la tecla con el número 8.

A continuación pulse 2 ó 3 veces SPACE y pulse nuevamente las mismas teclas 5, 6, 2 y 8, pero apretando simultáneamente CAPS SHIFT.

Como comprobará, los gráficos que aparecen en pantalla son exactamente los inversos a los que pulsó anteriormente.

3-2	MANEJANDO LOS CARACTERES GRAFICOS PREDEFINIDOS

En el programa del ejemplo anterior, podríamos hacer un marco más "bonito" con este desarrollo:

```
10 FOR i=0 TO 31
20 PRINT "▘ ";
30 NEXT i
40 FOR i=1 TO 20
50 PRINT "▘ ";TAB (31);"▘ "
60 NEXT i
70 FOR i=0 TO 31
80 PRINT "▚";
90 NEXT i
```



En otro orden de cosas es necesario saber cómo transformar números en caracteres. Cuando nosotros decimos que la tercera letra del alfabeto es la C, hemos aprovechado la relación existente entre la posición que ocupa en el alfabeto y la propia letra. Pues bien, algo similar ocurre con el conjunto de caracteres que maneja el Spectrum, gracias a la función CHR$.

Si nosotros tecleamos PRINT CHR$ (67) obtendremos el carácter que el Spectrum memoriza en la posición 67 dentro de su conjunto de caracteres (página 183 del manual).

Para ver en pantalla este conjunto corramos este programa.

```
10 FOR i=32 TO 255
20 PRINT CHR$ (i);
30 NEXT i
```

Con lo cual obtendremos:

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑_
£abcdefghijklmnopqrstuvwxyz{|}~©
 ▝▘▀▗▐▚▜▖▞▌▛▄▟▙█ABCDEFGHIJKLMNOP
QRSTURNDINKEY$PIFN POINT SCREEN$
 ATTR AT TAB VAL$ CODE VAL LEN S
IN COS TAN ASN ACS ATN LN EXP IN
T SQR SGN ABS PEEK IN USR STR$ C
HR$ NOT BIN OR AND <=>=<> LINE T
HEN TO STEP DEF FN CAT FORMAT MO
VE ERASE OPEN # CLOSE # MERGE VE
RIFY BEEP CIRCLE INK PAPER FLASH
 BRIGHT INVERSE OVER OUT LPRINT
LLIST STOP READ DATA RESTORE NEW
 BORDER CONTINUE DIM REM FOR GO
TO GO SUB INPUT LOAD LIST LET PA
USE NEXT POKE PRINT PLOT RUN SAV
E RANDOMIZE IF CLS DRAW CLEAR RE
TURN COPY
```

En realidad los 32 primeros códigos y los que siguen al 143 no son propiamente caracteres predefinidos y específicamente los 91 últimos corresponden a palabras sentencias del BASIC, que el Spectrum considera caracteres sin serlos.

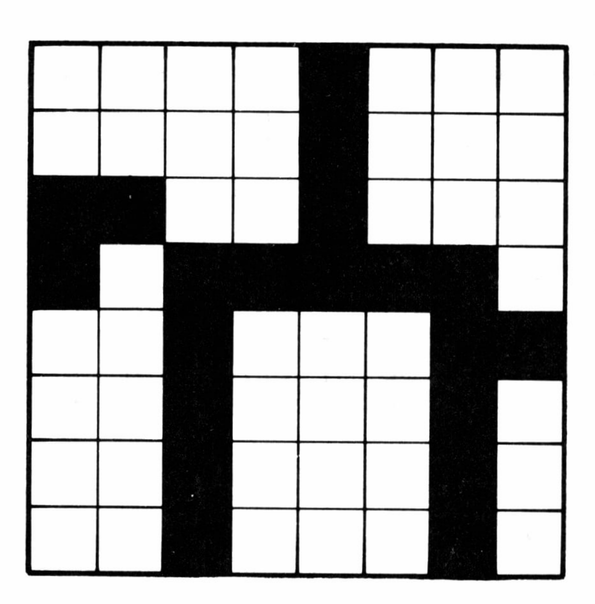
Hasta aquí lo que el Spectrum nos da por diseño, a partir de ahora veremos lo que nosotros podemos lograr de él.

3-3 CREANDO NUESTROS PROPIOS CARACTERES GRAFICOS

Supongamos que queremos crear un carácter gráfico que represente un jinete a caballo de tal forma que, siempre que apretemos la tecla adecuada y en modo gráfico, nuestro jinete aparezca.

En primer lugar no debemos olvidar que los caracteres —cualquier carácter, predefinido o no— deben estar comprendidos en una malla de 8 cuadritos horizontales por 8 verticales y que el carácter propiamente dicho, se logra con una adecuada combinación de cuadritos encendidos (color INK) y apagados (color PAPER).

Si representamos caballo y jinete en el cuadro correspondiente, tendremos:



Los cuadritos sombreados corresponden a los que hemos dado en llamar "encendidos", y que en lo que sigue los representaremos por 1 y los restantes son los apagados y que designamos por 0. Esto nos dará la siguiente combinación de 0 y 1:

Línea 0	0 0 0 0 1 0 0 0
Línea 1	0 0 0 0 1 0 0 0
Línea 2	1 1 0 0 1 0 0 0
Línea 3	1 0 1 1 1 1 1 0
Línea 4	0 0 1 0 0 0 1 1
Línea 5	0 0 1 0 0 0 1 0
Línea 6	0 0 1 0 0 0 1 0
Línea 7	0 0 1 0 0 0 1 0

La forma de crear este carácter, parte de cualquier letra desde la A a la U y por la sucesiva modificación de cada línea de 8 cuadritos horizontales de la malla de 8 x 8 donde está inscrito el carácter seleccionado. Por tanto, hay que elegir una letra, la A p.e., y comenzar modificando la línea 0 de la malla de 8 x 8, después la 1 y así hasta la 7.

Estas modificaciones se realizan con POKE USR y la última combinación de 0 y 1 en que acabamos de convertir el jinete y su caballo, de la siguiente manera:

```
10 POKE USR "a"+0,BIN 00001000
20 POKE USR "a"+1,BIN 00001000
30 POKE USR "a"+2,BIN 11001000
40 POKE USR "a"+3,BIN 10111110
50 POKE USR "a"+4,BIN 00111111
60 POKE USR "a"+5,BIN 00101010
70 POKE USR "a"+6,BIN 00100010
80 POKE USR "a"+7,BIN 00100010
```

Una vez introducida en la máquina esta rutina, y hasta que no se cambie la definición del carácter o se desconecte el ordenador, siempre que se apriete la tecla A en modo gráfico (cursor en G) aparecerá este jinete a caballo:



El proceso para ver en pantalla este carácter es:

1°.— Apretar PRINT.

2°.— Pasar el cursor a modo gráfico (CAPS SHIFT y GRAPHICS).

3°.— Apretar la tecla A.

En estas condiciones —con la rutina de nuestro jinete en la tecla A— corra el siguiente programa:

```
10 FOR i=0 TO 31
20 PRINT AT 0,31-i;"[graphic] ";
30 FOR j=0 TO 15: NEXT j
40 NEXT i
```

Supongamos que nos disponemos a escribir una carta. Lo primera que haremos, lógicamente, será proveernos de papel y lápiz.

Algo similar sucede con el Spectrum.

Con las instrucciones **INK** (TINTA) y **PAPER** (PAPEL) definimos el color de la tinta y el papel sobre el que vamos a escribir y para todo lo que siga a esos comandos, pero no a lo ya existente en la pantalla.

Ejemplo:

```
  10 INPUT "introduzca el numero
 del color del papel";p: PAPER p
  20 CLS
  30 INPUT "introduzca el numero
 del color de la tinta";t: INK t
  40 CLS
  50 PRINT "Los colores del Spec
trum"
  60 GO TO 10
```

En cambio, si INK y PAPER están contenidos en una instrucción que se inicie con la sentencia PRINT, estos colores son mantenidos hasta el momento en que la instrucción ha sido ejecutada, volviendo INK y PAPER a los colores que tuvieran anteriormente.

Compare el siguiente ejemplo con el anterior:

```
  10 INPUT "introduzca el numero
 del color del papel";p
  20 CLS
  30 INPUT "introduzca el numero
 del color de la tinta";t
  40 CLS
  50 PRINT INK t; PAPER p;"Los c
olores"; INK p; PAPER t;" del Sp
ectrum"
  60 GO TO 10
```

Observe cómo se comporta el color de fondo de la pantalla en ambos casos.

Con el comando CLS se "apagan" todos los cuadritos de la pantalla, con lo cual todo se torna en el color de PAPER, dando como resultado un borrado de la pantalla.

Con BORDER coloreamos toda la zona que circunvala el área de trabajo del Spectrum. En otras palabras, con BORDER se da color a todo lo que está fuera de los 45.086 cuadritos que controla el ordenador.

Ejemplo:

```
 10 INPUT "numero del color del
borde?";b: CLS
 30 BORDER b
 40 GO TO 10
```

Observe cómo el color de BORDER afecta también a las dos líneas últimas de la pantalla; para ello pruebe un BORDER negro y así forzará al ordenador a imprimir sus mensajes con tinta blanca.

Con INVERSE 1 conseguimos, para aquellos caracteres que sigan a esta instrucción, que todos los cuadritos cuya situación normal con respecto a esos caracteres debían estar encendidos, se apaguen y los que debían estar apagados se enciendan.

Con INVERSE 0 volvemos a la combinación de cuadritos apagados y encendidos estándar, para los caracteres afectados.

Ejemplo:

```
  10 INPUT "situacion de INVERSE
?";i: INVERSE i
  20 PRINT "Los colores del Spec
trum"
  50 GO TO 10
```

Antes de estudiar la sentencia OVER 1, recordemos que al imprimir un carácter en una posición de pantalla donde hay otro carácter, este último desaparece para dar lugar al nuevo.

Ejemplo:

```
10 PRINT AT 10,15;"/"
20 PRINT AT 10,15;"\"
```

Al correr este programa finalmente nos quedará en la pantalla el símbolo \.

Si añadimos:

5 CLS
7 OVER 1

nos quedará:

```
 5 CLS
 7 OVER 1
10 PRINT AT 10,15;"/"
20 PRINT AT 10,15;"\"
```

Corriendo este nuevo programa, resultará que la impresión en pantalla estará compuesta por dos barras que se cruzan y con el punto de corte de ambas en color PAPER.

Esta sentencia tal vez parezca complicada pero en definitiva trabaja de la siguiente manera: si dos caracteres coinciden en una misma posición de pantalla, la sentencia OVER 1 haría que todos los cuadritos apagados se mantengan apagados, mientras que los cuadritos encendidos del último carácter continúen encendidos si coinciden con cuadritos apagados del primer carácter, o se apaguen si coinciden con cuadritos encendidos.

Para anular la condición de sobreimpresión basta con la instrucción OVER 0.

La forma de conseguir un efecto de parpadeo partiendo de las características de sobreimpresión de OVER ya estudiadas, se basa en una rutina de este tipo:

```
  10 PRINT AT 10,3;"Los colores
del Spectrum"
  20 OVER 1
  30 FOR x=0 TO 100: NEXT x
  40 GO TO 10
```

Con BRIGHT 1 todos los cuadritos encendidos de los caracteres que sigan a esta instrucción aparecerán sobreiluminados, hasta que un BRIGHT 0 cancele esta situación.

Con FLASH 1 todos los cuadritos encendidos de los caracteres que sigan a esta instrucción, se apagarán y encenderán intermitentemente, produciendo un efecto de parpadeo hasta que un FLASH 0 cancele esta situación.

Estos dos últimos comandos pueden trabajar juntos para obtener una situación de parpadeo y sobreiluminación.

Ejemplo:

```
   5 INK 1: PAPER 7
  10 PRINT AT 0,3;"Los colores d
el Spectrum"
  12 PRINT AT 1,3;"NORMAL"
  15 PAUSE 100
  20 BRIGHT 1
  25 PRINT AT 5,3;"Los colores d
el Spectrum"
  27 PRINT AT 6,3;"SOBREBRILLO"
  30 PAUSE 100
  35 BRIGHT 0: FLASH 1
  40 PRINT AT 10,3;"Los colores
del Spectrum"
  42 PRINT AT 11,3;"PARPADEO"
  45 PAUSE 100
  50 BRIGHT 1
  55 PRINT AT 15,3;"Los colores
del Spectrum"
  60 PRINT AT 16,3;"SOBREBRILLO
Y PARPADEO"
  90 FLASH 0: BRIGHT 0
```

De la misma forma en que INK y PAPER trabajan dentro de una instrucción PRINT dando color sólo a los caracteres de esa instrucción, también responderán INVERSE, OVER, FLASH y BRIGHT.

Ejemplo:

```
  10 PRINT "Los colores"
  20 PRINT BRIGHT 1;AT 0,12;"del
Spectrum"
```

Un resumen de lo aquí tratado y unas consideraciones de carácter general darán por finalizada esta primera parte.

El Spectrum maneja la pantalla del televisor considerando la parte central como una malla de 256 cuadritos horizontales por 176 verticales. El color de esta zona se controla con la instrucción PAPER. El color del resto de la pantalla, que rodea a la anterior, se controla con BORDER.

Inicialmente los colores de estas dos áreas son blancos. Los caracteres se imprimen en la zona central de la pantalla y sobre el color fijado por el comando PAPER. La figura de los caracteres se obtiene sobre mallas de 8 x 8 cuadritos, lo cual da lugar a un máximo de 22 líneas de 32 caracteres cada una, es decir 704 (22 x 32) posiciones posibles de carácter.

Hay dos líneas inferiores que quedan reservadas para la entrada de datos y la emisión de mensajes por parte de la máquina.

Cada malla de carácter de 8 x 8 cuadritos tiene unas características propias que son conocidas como sus **atributos.** Estos atributos son cuatro:

PAPER.– Es el color de los cuadritos que rodean la propia figura del carácter y que hemos considerado como apagados. En otras palabras, con PAPER definimos el color de fondo de la malla de 8 x 8. Como ya se vio, su color inicial es blanco.

INK.– Es el color de los cuadritos que definen la figura del carácter y que hemos considerado como encendidos. Su color inicial es negro.

Cuando conectamos el Spectrum escribimos en negro sobre fondo blanco.

BRIGHT.– Cada malla de 8 x 8 cuadritos puede mostrar su contenido con un brillo normal, como sucede al conectar la máquina, o con sobrebrillo, lo cual se obtiene con BRIGHT 1.

FLASH.– Cada malla de 8 x 8 cuadritos puede mostrar su contenido manteniendo el color de los cuadritos en sus colores INK y PAPER o, gracias a FLASH 1, convirtiendo alternativamente los cuadritos en color INK, en cuadritos color PAPER y al revés.

Quede claro pues, que estos atributos están referidos exclusivamente a cada malla de carácter, por lo cual no podemos manipular cada cuadrito que forma la malla de 8 x 8 que la componen. No obstante, sí podemos asignar atributos a cada una de las 768 posiciones de carácter (24 líneas x 32 columnas, si consideramos las dos líneas últimas reservadas a los mensajes del ordenador) los cuales quedarán guardados en el "**área de atributos** de la memoria RAM".

Como se puede ver en la página 165 del manual, el **"área de atributos"** del mapa de memoria permite almacenar 768 bytes; un byte por cada posición de carácter.

La forma en que cada byte de esta zona de memoria controla los atributos de cada posición de carácter es la siguiente: los tres primeros bits (del 0 al 2) controlan el color de INK, los tres siguientes (del 3 al 5) el color de PAPER, el siguiente (6) el BRIGHT y el último (7) controla el parpadeo (FLASH). Más tarde volveremos sobre esto, y especialmente sobre cómo utilizar la función ATTR.

Todos los atributos pueden ser usados, como ya vimos, dentro de una instrucción PRINT, en cuyo caso son definidos como **atributos temporales** ya que, una vez ejecutada la instrucción, los atributos vuelven a las condiciones que imperaban con anterioridad. Por el contrario, cuando los atributos son fijados mediante instrucciones cuya primera palabra sea una de las correspondientes a los atributos, entonces los atributos son considerados como **permanentes.**

En el caso de que deseemos efectuar una impresión sobre una o varias posiciones de carácter, pero sin alterar todos o algunos de los atributos que tengan fijados, bastará con usar el número 8. Así, si queremos realizar una impresión que mantenga la misma situación de brillo (normal o sobreiluminada) que tuviera esa posición de carácter, sólo deberemos introducir BRIGHT 8. Claro es que, si conocemos que en esa posición (o posiciones) de carácter hay una situación de sobreiluminación, sólo tendríamos que haber usado BRIGHT 1. En otras palabras, usaremos el 8 cuando no conozcamos la situación que afecta a los atributos de esa posición de carácter y se quieran mantener.

En otro orden de cosas puede ocurrir que, sin conocer el color de PAPER del que actualmente sea el fondo, se necesite un color de INK con suficiente contraste como para poder ser leído. En este momento se puede utilizar el color INK 9- con lo cual obtendremos un color de INK que, al menos, nos deje entender lo impreso. De igual forma, si no conocemos el color de INK y necesitamos un fondo que nos dé suficiente contraste, usaremos PAPER 9. Como el color 9 lo suministra el computador de acuerdo con las anteriores premisas, no siempre se obtendrá el mejor contraste pero sí, al menos, el mínimo necesario.

LA CASUALIDAD

Nada más lejos de la casualidad que un computador. En él, todo está previsto. No obstante para multitud de aplicaciones es absolutamente necesario "manejar la casualidad". Introducir la incertidumbre.

Cuando tiramos una moneda al aire, existen dos posibilidades: obtener una cara o una cruz. De antemano no sabemos qué va a caer. Algo similar sucede en el lanzamiento de dados o el girar de una ruleta.

Cabe pensar que con un computador no es posible llegar a situaciones imprevistas, dado que sólo se pueden obtener resultados partiendo de funciones preestablecidas.

En sentido estricto esto es cierto. Con un computador no se pueden pretender situaciones de puro azar. Todo debe estar sometido a las matemáticas. No obstante, apoyándonos en ellas conseguiremos números impredecibles desde el punto de vista práctico, los cuales son calculados a partir de una fórmula compleja y con tal rapidez, que es imposible alcanzar el resultado antes que la máquina.

Estos números que no son puramente casuales se denominan **pseudoaleatorios** y son los que producen los ordenadores a través de la función **RND**.

Con RND obtendremos números iguales o mayores que 0 y menores que 1 pero, con unos sencillos artificios, podemos obtener cualquier otro rango de variaciones. Por ejemplo:

Números comprendidos entre 1 y 10 se obtienen aleatoriamente multiplicando RND por 10, extrayendo su parte entera y sumándole uno:

```
PRINT INT (RND * 10) + 1
```

RND produce la misma serie de respuestas cada vez que se conecta la máquina. Esta característica puede ser útil para contrastar el comportamiento de distintos modelos ante la misma gama de posibilidades.

Para conseguir una sensación de casualidad total el BASIC nos provee de la sentencia **RAND** (abreviación de RANDOMIZE).

La función RAND actúa como puntero, indicando a RND donde debe iniciar su serie.

Con RAND 0 se obtiene la máxima "casualidad" posible, ya que RND iniciará su serie de acuerdo con el tiempo que lleve conectado el Spectrum. En función de esto, la rutina que nos proveerá de resultados aleatorios será:

```
10 RANDOMIZE 0
20 PRINT RND
30 GO TO 20
```

Si quisiéramos jugar a "cara o cruz" con una moneda o, lo que es igual, estudiar cómo se agrupan los resultados de un suceso dicotómico, podríamos aplicar un programa de este tipo:

```
  10 RANDOMIZE 0
  20 LET x=INT (RND*2+1)
  30 IF x=1 THEN PRINT "caras":
GO TO 20
  40 PRINT "cruces"
  50 GO TO 20
```

En el caso que se deseen analizar los resultados producidos en una serie de lanzamientos de un dado se podría utilizar una rutina como la que sigue:

```
10 RANDOMIZE 0
20 LET x=INT (RND*6+1)
30 PRINT x
40 GO TO 20
```

Este mismo caso, pero aplicado a un juego, requeriría algo de movimiento y, desde luego, más emoción. Probemos este otro:

```
   2 PAPER 6: INK 1: BORDER 1
   5 GO TO 340
  10 PRINT AT 0,0;"           ";AT 1
,0;"           ";AT 2,0;"           ";AT
 3,0;"    ■     ";AT 4,0;"           ";
AT 5,0;"           ";AT 6,0;"
"
  15 RETURN
  20 GO SUB 10
  25 PRINT AT 3,0;" ■   ■ "
  26 RETURN
  30 GO SUB 20
  35 PRINT AT 3,3;"■"
  37 RETURN
  40 GO SUB 10
  45 PRINT AT 1,0;" ■   ■ ";AT 5
,0;" ■   ■ ";AT 3,3;" "
  46 RETURN
```

```
 50 GO SUB 40
 55 PRINT AT 3,3;"■"
 56 RETURN
 60 GO SUB 50
 65 PRINT AT 1,3;"■";AT 5,3;"■"
;AT 3,3;" "
 66 RETURN
260 RANDOMIZE 0
270 LET x=INT (RND*6+1)*10
280 FOR i=1 TO 6
290 GO SUB i*10
300 FOR j=1 TO i
305 BEEP .01,1
310 NEXT j
320 NEXT i
330 GO SUB x
340 INPUT "otra tirada? s/n";a$
350 IF a$<>"s" THEN PRINT AT 19
,0;"hasta pronto": STOP
360 GO TO 270
```

En este programa es de notar la interrelación existente entre los números de las líneas de programa y las subrutinas para generar las caras de un imaginario dado que nos daría una impresión de este tipo:

```
■   ■
  ■
■   ■
```

Así, la línea 10 nos dibuja un carácter negro para representar la cara del "uno", la línea 20 la cara del "dos", la línea 30 la del "tres" y así hasta la línea 60, que nos dará la cara del "seis". En las líneas 260 y 270 se genera un número aleatorio comprendido entre 10 y 60 que, con la instrucción de la línea 330, nos está dando los valores del dado en cada jugada.

Finalmente, en la rutina comprendida entre 280 y 320, se simula la caída del dado en el sentido de mostrarnos varias caras del dado antes de darnos la definitiva y, de esta forma, darle suspense al juego.

Observe que algunas caras del dado no reproducen exactamente las caras de un dado normal. Lo hemos hecho así para no complicar excesivamente el programa.

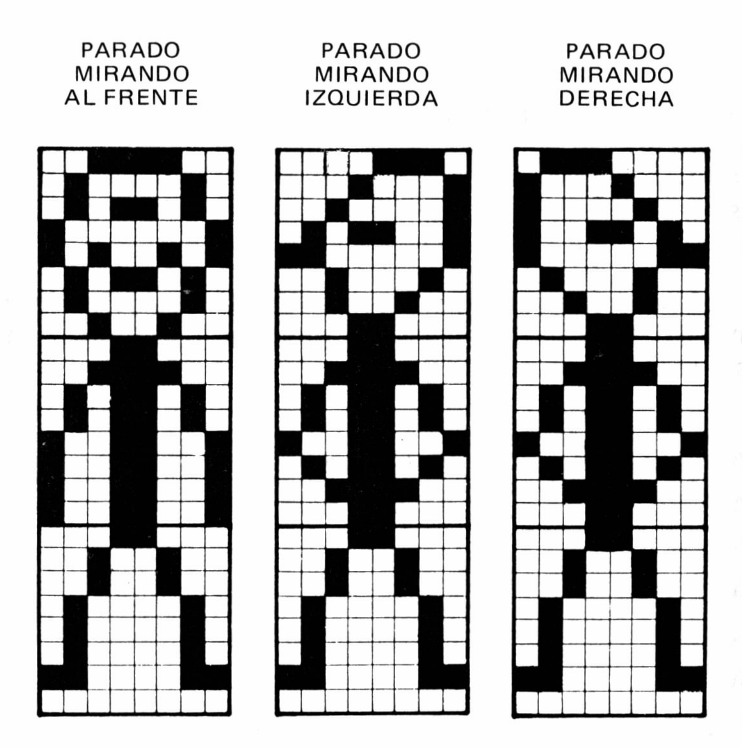
El título de este epígrafe es menos extraño de lo que parece. D. Pepe es el simpático protagonista de nuestras aventuras y, a semejanza de Geppeto y su Pinocho, vamos a darle "vida". En la medida que lo consigamos habremos dominado el movimiento de gráficos en pantalla.

En primer lugar hay que saber que el ánima de D. Pepe, como el de cualquier trabajo que realicemos con el computador, es un trozo de nuestra imaginación.

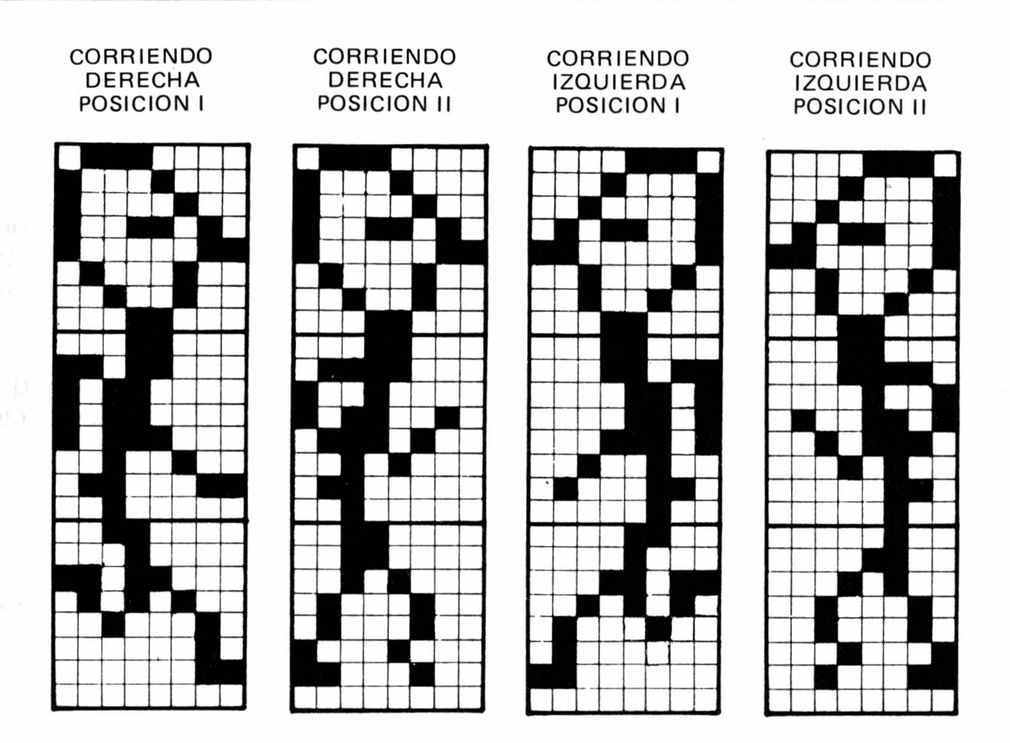
Cuanto más clara sea la imagen mental de lo que pretendemos hacer, tanto más fácil nos será llevarlo a la práctica; así pues empecemos por describir a D. Pepe y su "papel" en la pantalla.

D. Pepe es un muñeco que tiene cabeza, tronco y extremidades y su "trabajo" consistirá en correr a derecha, izquierda y "observar".

Hagamos unos bocetos de D. Pepe:



3-6 DON PEPE O EL MOVIMIENTO



Cuando D. Pepe esté parado moverá la cabeza a derecha, al frente y a la izquierda, adoptando las posiciones de cabeza de "corriendo a la derecha" y "corriendo a la izquierda" alternativamente y extendiendo los brazos, según muestran las zonas sombreadas de los gráficos correspondientes. Como se ve, nuestro muñeco ocupará tres posiciones verticales de carácter en pantalla.

Según lo aprendido anteriormente sobre los gráficos definidos por el usuario, vamos a asimilar las siguientes teclas, en modo gráfico (cursor en G), a diferentes "trocitos" de D. Pepe:

TECLA A (A) .... Cabeza mirando de frente
TECLA B (B) .... Cabeza mirando a izquierda
TECLA C (C) .... Cabeza mirando a derecha
TECLA D (D) .... Cuerpo brazos extendidos
TECLA E (E) .... Cuerpo brazos en jarras
TECLA F (F) .... Piernas parado
TECLA G (G) .... Piernas corriendo derecna-Posición I

TECLA H (H) . . . . Piernas corriendo izquierda-Posición I
TECLA I (I) . . . . Piernas corriendo derecha-Posición II
TECLA J (J) . . . . Piernas corriendo izquierda-Posición II
TECLA K (K) . . . . Cuerpo corriendo derecha-Posición I
TECLA L (L) . . . . Cuerpo corriendo derecha-Posición II
TECLA M (M) . . . . Cuerpo corriendo izquierda-Posición I
TECLA N (N) . . . . Cuerpo corriendo izquierda-Posición II

Las rutinas que nos darían las posiciones de D. Pepe, según lo ya estudiado, serían las siguientes:

```
   9 REM ***cabeza mirando de fr
ente***
  10 POKE USR "a"+0,BIN 00111100
  20 POKE USR "a"+1,BIN 01000010
  30 POKE USR "a"+2,BIN 01011010
  40 POKE USR "a"+3,BIN 10000001
  50 POKE USR "a"+4,BIN 10100101
  60 POKE USR "a"+5,BIN 01011010
  70 POKE USR "a"+6,BIN 01000010
  80 POKE USR "a"+7,BIN 00100100
  99 REM ***cabeza mirando a la
izquierda***
 100 POKE USR "b"+0,BIN 00001110
 110 POKE USR "b"+1,BIN 00010001
 120 POKE USR "b"+2,BIN 00100001
 130 POKE USR "b"+3,BIN 01011001
 140 POKE USR "b"+4,BIN 11000001
 150 POKE USR "b"+5,BIN 00100010
 160 POKE USR "b"+6,BIN 00100100
 170 POKE USR "b"+7,BIN 00011000
 199 REM ***cabeza mirando a la
derecha***
 200 POKE USR "c"+0,BIN 01110000
 210 POKE USR "c"+1,BIN 10001000
 220 POKE USR "c"+2,BIN 10000100
 230 POKE USR "c"+3,BIN 10011010
 240 POKE USR "c"+4,BIN 10000011
 250 POKE USR "c"+5,BIN 01000100
 260 POKE USR "c"+6,BIN 00100100
 270 POKE USR "c"+7,BIN 00011000
 299 REM ***cuerpo brazos extend
idos***
 300 POKE USR "d"+0,BIN 00011000
 310 POKE USR "d"+1,BIN 00111100
 320 POKE USR "d"+2,BIN 01011010
 330 POKE USR "d"+3,BIN 01011010
```

```
 340 POKE USR "d"+4,BIN 10011001
 350 POKE USR "d"+5,BIN 10011001
 360 POKE USR "d"+6,BIN 10011001
 370 POKE USR "d"+7,BIN 10011001
 399 REM ***cuerpo brazos en jar
ras***
 400 POKE USR "e"+0,BIN 00011000
 410 POKE USR "e"+1,BIN 00111100
 420 POKE USR "e"+2,BIN 01011010
 430 POKE USR "e"+3,BIN 01011010
 440 POKE USR "e"+4,BIN 10011001
 450 POKE USR "e"+5,BIN 01011010
 460 POKE USR "e"+6,BIN 00111100
 470 POKE USR "e"+7,BIN 00011000
 499 REM ***piernas parado***
 500 POKE USR "f"+0,BIN 00011000
 510 POKE USR "f"+1,BIN 00100100
 520 POKE USR "f"+2,BIN 00100100
 530 POKE USR "f"+3,BIN 01000010
 540 POKE USR "f"+4,BIN 01000010
 550 POKE USR "f"+5,BIN 01000010
 560 POKE USR "f"+6,BIN 11000011
 570 POKE USR "f"+7,BIN 00000000
 599 REM ***piernas corriendo de
recha posicion I***
 600 POKE USR "g"+0,BIN 00110000
 610 POKE USR "g"+1,BIN 00013000
 620 POKE USR "g"+2,BIN 11011000
 630 POKE USR "g"+3,BIN 01010100
 640 POKE USR "g"+4,BIN 00100010
 650 POKE USR "g"+5,BIN 00000010
 660 POKE USR "g"+6,BIN 00000011
 670 POKE USR "g"+7,BIN 00000000
 699 REM ***piernas corriendo iz
quierda posicion I***
 700 POKE USR "h"+0,BIN 00001100
 710 POKE USR "h"+1,BIN 00001000
 720 POKE USR "h"+2,BIN 00011011
 730 POKE USR "h"+3,BIN 00101010
 740 POKE USR "h"+4,BIN 01000100
 750 POKE USR "h"+5,BIN 01000000
 760 POKE USR "h"+6,BIN 11000000
 770 POKE USR "h"+7,BIN 00000000
 799 REM ***piernas corriendo de
recha posicion II***
 800 POKE USR "i"+0,BIN 00100000
 810 POKE USR "i"+1,BIN 00110000
 820 POKE USR "i"+2,BIN 00101000
 830 POKE USR "i"+3,BIN 01000100
 840 POKE USR "i"+4,BIN 01000100
 850 POKE USR "i"+5,BIN 10001000
 860 POKE USR "i"+6,BIN 11000100
 870 POKE USR "i"+7,BIN 00000000
 899 REM ***piernas corriendo iz
quierda posicion II***
```

```
 900 POKE USR "j"+0,BIN 00000100
 910 POKE USR "j"+1,BIN 00001100
 920 POKE USR "j"+2,BIN 00010100
 930 POKE USR "j"+3,BIN 00100010
 940 POKE USR "j"+4,BIN 00100010
 950 POKE USR "j"+5,BIN 00010001
 960 POKE USR "j"+6,BIN 00100011
 970 POKE USR "j"+7,BIN 00000000
 999 REM ***cuerpo corriendo der
echa posicion I***
1000 POKE USR "k"+0,BIN 00011000
1010 POKE USR "k"+1,BIN 11011000
1020 POKE USR "k"+2,BIN 10110000
1030 POKE USR "k"+3,BIN 10110000
1040 POKE USR "k"+4,BIN 10111000
1050 POKE USR "k"+5,BIN 00100100
1060 POKE USR "k"+6,BIN 01100010
1070 POKE USR "k"+7,BIN 00100000
1099 REM ***cuerpo corriendo der
echa posicion II***
1100 POKE USR "l"+0,BIN 00011000
1110 POKE USR "l"+1,BIN 01111000
1120 POKE USR "l"+2,BIN 10010000
1130 POKE USR "l"+3,BIN 10110010
1140 POKE USR "l"+4,BIN 01110100
1150 POKE USR "l"+5,BIN 00101000
1160 POKE USR "l"+6,BIN 01100000
1170 POKE USR "l"+7,BIN 00100000
1199 REM ***cuerpo corriendo izq
uierda posicion I***
1200 POKE USR "m"+0,BIN 00011000
1210 POKE USR "m"+1,BIN 00011011
1220 POKE USR "m"+2,BIN 00001101
1230 POKE USR "m"+3,BIN 00001101
1240 POKE USR "m"+4,BIN 00011101
1250 POKE USR "m"+5,BIN 00100100
1260 POKE USR "m"+6,BIN 01000110
1270 POKE USR "m"+7,BIN 00000100
1299 REM ***cuerpo corriendo izq
uierda posicion II***
1300 POKE USR "n"+0,BIN 00011000
1310 POKE USR "n"+1,BIN 00011110
1320 POKE USR "n"+2,BIN 00001001
1330 POKE USR "n"+3,BIN 01001101
1340 POKE USR "n"+4,BIN 00101110
1350 POKE USR "n"+5,BIN 00010100
1360 POKE USR "n"+6,BIN 00000110
1370 POKE USR "n"+7,BIN 00000100
```

De esta forma, en las teclas convenidas, tendríamos esta serie de caracteres:



Los cuales, debidamente acoplados darán unas imágenes, como se verá más adelante, similares a éstas:



Vamos a analizar algunos conceptos básicos del movimiento antes de continuar.

Es práctico evitar, en lo posible, utilizar la alta resolución de gráficos —tema que veremos más adelante— ya que obligamos a la máquina a trabajar más, debiendo, por tanto, aprovechar todas las posibilidades que nos ofrecen los caracteres gráficos definidos —o no— por el usuario.

En otro orden de cosas sabemos que, tanto el cine como la televisión transmiten sus imágenes secuencialmente, de tal forma que nuestra vista lo interpreta como un movimiento continuo. Nosotros utilizaremos el mismo principio, apoyándonos en los medios que nos brinda el computador. Empecemos por hacer aparecer y desaparecer una imagen con una velocidad controlada:

```
1000 LET e=10
1010 LET a=60
1020 PRINT AT 0,0;"▚"
1030 FOR i=1 TO e: NEXT i
1040 PRINT AT 0,0;" "
1050 FOR j=1 TO a: NEXT j
1060 GO TO 1000
```

Las líneas 1000 y 1010 nos fijan tiempos de encendido **(e)** y apagado **(a)**. Los bucles 1030 y 1050 son retardos de encendido y apagado. En la línea 1040 colocamos un carácter blanco (con el cursor en modo gráfico apretar la tecla con el número 8) para producir la sensación de "apagado". Cambie Vd. los valores de **e** y **a** para observar diferentes posibilidades.

El siguiente paso para conseguir el efecto de movimiento, consistirá en hacer "aparecer" el carácter, hacerlo "desaparecer" en la misma posición, para volverlo hacer aparecer uno o varios espacios más adelante:

```
1000 PRINT AT 0,0;"'▚"
1010 FOR j=0 TO 10: NEXT j
1020 FOR i=0 TO 30
1030 FOR j=0 TO 10: NEXT j
1040 PRINT AT 0,i;" ▚"
1050 NEXT i
1060 GO TO 1000
```

En las líneas 1000 producimos la impresión de la primera posición. En la 1020 tenemos un bucle que obligará al gráfico a recorrer toda la línea 0.

En la línea 1040 hacemos avanzar el gráfico una posición "borrando" la anterior, gracias al carácter "blanco" introducido. Con esta rutina el movimiento se produce al margen de nuestro control.

Completemos esta introducción al movimiento, desarrollando un programa que nos permita controlar el desplazamiento de un hipotético muñeco que ocupa dos posiciones de pantalla, una encima de otra, tal como éste:



y que podría haber estado, sin dificultad, compuesto por dos "trocitos" de D. Pepe, pero que por razones prácticas está formado por un asterisco arriba y otros caracteres gráficos abajo.

Con este carácter combinado vamos a lograr la sensación de que la parte inferior está moviéndose constantemente:

3-6	DON PEPE O EL MOVIMIENTO

```
1000 LET c=15
1010 LET a$=INKEY$
1020 IF a$="5" THEN LET c=c-1
1030 IF a$="8" THEN LET c=c+1
1040 PRINT AT 0,c;"    *    ";AT 1
,c;"   ▛▟   ";
1050 BEEP .01,10
1060 PRINT AT 0,c;"    *    ";AT 1
,c;"   ▛▟   ";
1070 GO TO 1010
```

El comando BEEP, además de emitir un sonido, actúa como una pausa o bucle de retardo. Dicho de otro modo, durante el tiempo que está emitiendo un BEEP, el computador no hace otra cosa y esto se puede comprobar alargando la duración de la línea 1050 del anterior programa con un BEEP 4, 10, por ejemplo.

Las normas básicas del movimiento son las expuestas anteriormente y serán de aplicación inmediata en la animación de D. Pepe, con algunas variaciones. En primer lugar, los gráficos son distintos según se desplace nuestro amigo a la derecha o a la izquierda. En segundo lugar la cabeza ha de mirar, como ya se acordó más arriba, a derecha, al frente y a la izquierda cuando esté parado y extendiendo sus brazos —desde la posición en "jarras"— alternativamente.

Veamos este último caso:

```
2000 REM ***Don Pepe***
2010 LET c=15
2020 GO SUB 2060
2030 PRINT AT 0,c;" A ";AT 1,c;"
 E ";AT 2,c;" F ";
2040 BEEP .3,15
2050 PRINT AT 0,c;" B ";AT 1,c;"
 D ": BEEP .3,5
2051 PRINT AT 0,c;" A ";AT 1,c;"
 E ";AT 2,c;" F ": BEEP .3,15
2052 PRINT AT 0,c;" C ";AT 1,c;"
 D "
2060 BEEP .3,5
2070 GO TO 2020
```

Este programa no se puede correr.

Doy por supuesto, que las rutinas que generan los caracteres gráficos de los distintos "trocitos" de D. Pepe ya están en memoria.

El programa que nos determinará a D. Pepe corriendo a izquierda o derecha y nos permitirá controlar el movimiento, se basa en todo lo estudiado anteriormente:

```
2080 LET a$=INKEY$
2090 IF a$="5" THEN LET c=c-1: G
O SUB 2120: GO TO 2080
2100 IF a$="8" THEN LET c=c+1: G
O SUB 2180: GO TO 2080
2110 RETURN
2120 IF c<0 THEN LET c=30: PRINT
 AT 0,1;"  ";AT 1,1;"  ";AT 2,1;"
"
2130 PRINT AT 0,c;" B ";AT 1,c;"
 M ";AT 2,c;" H "
2140 BEEP .1,15
2150 PRINT AT 0,c;" B ";AT 1,c;"
 N ";AT 2,c;" J "
2160 BEEP .1,20
2170 RETURN
2180 IF c>30 THEN LET c=0: PRINT
 AT 0,31;"  ";AT 1,31;"  ";AT 2,31
;"  "
2190 PRINT AT 0,c;" C";AT 1,c;"
K";AT 2,c;" G"
2200 BEEP .1,15
2210 PRINT AT 0,c;" C ";AT 1,c;"
 L ";AT 2,c;" I "
2220 BEEP .1,20
2230 RETURN
```

Este programa no se puede correr tal y como está. Para conseguir el efecto de movimiento de Don Pepe, será necesario correr juntos los programas 22, 26 y 27.

A otro nivel de complejidad se llega al introducir más de un movimiento simultáneo.

Los rudimentos de estos programas se basan en la lectura continua y secuencial de las condiciones que se le impongan a cada movimiento.

Para ver esto supongamos dos asteriscos que se han de mover por la misma línea y con una velocidad, de uno con respecto a otro, del doble. Esta última condición implica que mientras un asterisco avanza un espacio el otro avanza dos.

Veamos este ejemplo:

```
10 LET c=0
20 LET c1=0
30 LET v=1: LET v1=2
40 GO SUB 70
50 GO SUB 120
60 GO TO 40
70 PRINT AT 0,c;" "
80 LET c=c+v
```

```
 90 IF c=31 OR c=0 THEN LET v=-
v
100 PRINT AT 0,c;"*"
110 RETURN
120 PRINT AT 5,c1;" "
130 LET c1=c1+v1
140 IF c1>31 THEN LET c1=0
150 PRINT AT 5,c1;"*"
160 RETURN
```

En la línea 30 fijamos las velocidades de cada asterisco. En la 40 dirigimos el programa a la subrutina de avance del más lento. En la 50 a la de avance del más rápido. En la línea 90 invertimos la marcha del asterisco más lento al llegar a los límites de la pantalla.

En la 140 determinamos que el asterisco más rápido al llegar al límite izquierdo de la pantalla, vuelva a comenzar.

Gracias la línea 90 vemos "rebotar" el asterisco en los bordes izquierdo y derecho de la pantalla, moviéndose siempre en la misma línea.

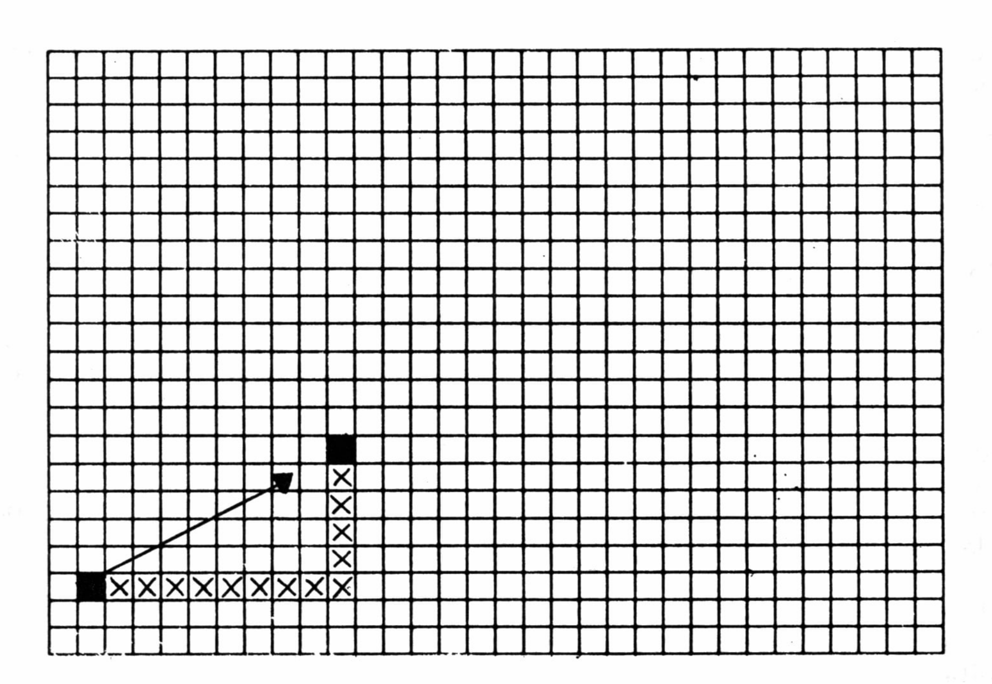
En el caso de balones o pelotas rebotando contra paredes, las trayectorias ya no tienen por qué ser horizontales o verticales y, por tanto, requieren un tratamiento especial.

Una pelota en movimiento de un punto cualquiera a otro y sometida a una velocidad **(v)**, se puede descomponer en una velocidad horizontal **(h)** y en una vertical **(v)**. En otras palabras, cada desplazamiento de la pelota requiere moverla un número de espacios horizontales y otro (igual o distinto) verticales. (Ver figura en página **41**).

Si llamamos **l** a la coordenada que representa a las líneas y **c** a la que representa a las columnas, deberemos añadir la velocidad horizontal (h) a la coordenada **c** y la velocidad vertical (v) a la **l**.

Por otra parte resulta evidente que, cuando una pelota "choca" con una pared vertical, devuelve la velocidad horizontal en sentido contrario, ya que la vertical queda invariable.

Cuando "choca" con una pared horizontal, por similar razón, sólo devuelve la vertical.



Si aplicamos estos conceptos veremos a la pelota rebotar, ausente de efectos tales como el rozamiento, la gravedad, etc. Estas cuestiones se verán más adelante.

De momento estudiemos este programa:

```
  5 GO SUB 80
 10 LET l=1: LET c=1
 20 LET v=1: LET h=1
 30 PRINT AT l,c;" "
 40 LET l=l+v: LET c=c+h
 50 IF l=1 OR l=20 THEN LET v=-
v: BEEP .3,15
 60 IF c=1 OR c=30 THEN LET h=-
h: BEEP .3,20
 70 PRINT AT l,c;"■"
 72 PAUSE 5
 75 GO TO 30
 80 FOR i=0 TO 31
 90 PRINT AT 0,i;"■";
100 PRINT AT 21,i;"■";
110 IF i>21 THEN GO TO 140
120 PRINT AT i,0;"■";
130 PRINT AT i,31;"■";
140 NEXT i
150 RETURN
```

Con el bucle que se inicia en la línea 80, dibujamos el marco de la zona de "rebotes".

En las líneas 10 y 20 determinamos las coordenadas iniciales y las velocidades.

En la 40 varían las coordenadas de acuerdo con las velocidades.

Con las líneas 50 y 60 se fijan las condiciones de rebote.

En BASIC, a medida que aumentamos el número de movimientos que deban aparecer como simúltaneos, la velocidad de respuesta baja considerablemente. No obstante vamos a estudiar un caso de tres movimientos posibles simultáneos.

Supongamos la pantalla como un recinto cerrado y que a lo largo de sus paredes izquierda y derecha se deslizan sendas raquetas, las que, controladas por teclado, deberán impedir que una pelota, moviéndose por la pantalla, choque con las paredes que tratan de proteger las raquetas.

El siguiente programa nos da una idea de lo que puede ser un juego de tenis o, en términos de lo que pretendemos, un movimiento triple y, aparentemente, simultáneo:

```
1,0: RETURN
 220 LET h=-h
 230 RETURN
 300 LET dI=l-lri
 310 IF dI<0 OR dI>5 THEN BEEP .
1,0: RETURN
 320 LET h=-h
 330 RETURN
 340 REM ********************
 400 LET a$=INKEY$
 410 IF a$="0" AND lrd>1 THEN LE
T lrd=lrd-1: GO TO 430
 420 IF a$="p" AND lrd<17 THEN L
ET lrd=lrd+1
 430 PRINT AT lrd,31;" ";AT lrd+
1,31;"█";AT lrd+2,31;"█";AT lrd+
3,31;"█";AT lrd+4,31;" "
 440 RETURN
 500 LET b$=INKEY$
 510 IF b$="1" AND lri>1 THEN LE
T lri=lri-1: GO TO 530
 520 IF b$="q" AND lri<17 THEN L
ET lri=lri+1
 530 PRINT AT lri,0;" ";AT lri+1
,0;"█";AT lri+2,0;"█";AT lri+3,0
;"█";AT lri+4,0;" "
 540 RETURN
```

```
  2 BORDER 1: INK 1: PAPER 6
  5 REM ***recuadro de juego***
 10 FOR i=0 TO 31
 20 PRINT INK 7;AT 0,i;"█";AT 2
1,i;"█"
 50 NEXT i
 52 PAUSE 50
 55 REM **coordenadas iniciales
 de la pelota***
 60 LET l=5: LET c=15
 65 REM ***velocidad inicial de
 la pelota***
 70 LET v=1: LET h=1
 75 REM ***coordenadas de raque
ta derecha***
 80 LET lrd=10: LET crd=31
 85 REM ***coordenadas de raque
ta izquierda***
 90 LET lri=10: LET cri=0
 95 REM ***trayectoria pelota**
*
100 GO SUB 130
105 REM ***desplazamiento raque
ta derecha***
110 GO SUB 400
115 REM ***desplazamiento raque
ta izquierda***
120 GO SUB 500
122 REM ***cierra el proceso y
lo repite***
125 GO SUB 100
130 IF l=1 OR l=20 THEN LET v=-
v: BEEP .3,15
135 IF c=0 OR c=31 THEN CLS : G
O TO 5
140 IF c=1 THEN GO SUB 300
150 IF c=30 THEN GO SUB 200
160 PRINT INK 1;AT l,c;" "
170 LET l=l+v: LET c=c+h
180 PRINT AT l,c;"█"
190 RETURN
195 REM ******************
200 LET dD=l-lrd
210 IF dD<0 OR dD>5 THEN BEEP .
```

Este programa requiere pocos comentarios, ya que los REM introducidos, junto con todo lo visto hasta el momento, lo hacen innecesario. Analice el listado del programa y, además de aumentar su capacidad de análisis, averiguará cómo manejar las raquetas, el movimiento de la pelota, etc.

3-6	DON PEPE O EL MOVIMIENTO

Unas consideraciones finales ayudarán en el diseño de programas. En primer lugar la programación en BASIC, como se ha podido ver, no presenta especial dificultad pero, a veces, los resultados obtenidos aparecen como lentos debido a que el Spectrum traduce a lenguaje máquina, una a una, todas las instrucciones de un programa; este tipo de "traducción" hace que llamemos **intérprete** a esta forma de BASIC, en contraposición al **compilador,** el cual traduce el programa completo a lenguaje máquina antes de correrlo dando, por esto, una velocidad de proceso mucho mayor.

Otro motivo de "lentitud" procede de los saltos incondicionales tales como GOTO o GOSUB, que obligan a continuar el programa en una línea determinada, pero, como la localización de esta línea implica una búsqueda secuencial desde el principio, siempre hay una pérdida añadida de tiempo. De aquí debe seguirse que una adecuada ubicación de subrutinas y funciones ayudará a mejorar la velocidad de ejecución de los programas.

En todo caso, antes de llegar a diseñar un programa que sea realmente operativo, habrá que hacer modificaciones y solventar errores.

Cuando aparece un error en un programa bien estructurado es relativamente fácil localizarlo y aplicar el remedio oportuno. Para conseguir una buena estructuración se debe tratar fundamentalmente:

1°.— Definir con claridad las variables y tratar de agruparlas por tipos. Por ejemplo: V, V1, V2, ... v, v1, v2, para referirnos a las velocidades.

2°.— Colocar los bucles FOR/NEXT en una sola línea. Si tal cosa no fuera posible, entonces colocar las sentencias FOR y NEXT al principio de su línea, de forma que sean fácilmente identificables.

3°.— Las sentencias REM son muy útiles para clarificar un programa, pero no conviene abusar de ellas pues restan velocidad.

4°.— Siempre que pueda escriba programas cortos que recurran a tantas subrutinas como necesite, de forma que los errores y modificaciones sean fáciles de poner en orden. Obviamente, aquí las subrutinas tienen una misión clarificadora y no, necesariamente, son introducidas porque vayan a ser utilizadas muchas veces.

5°.— Numerar el programa de acuerdo con grupos de actividades. Por ejemplo: hasta la línea 500 fijar parámetros, de la 1000 a la 2000 colocar el programa principal, etc.

Una vez estructurado el programa, si éste —al hacerlo correr— no funciona adecuadamente, deberá ser comprobado para encontrar los motivos. Algunas de las normas básicas del proceso de comprobación pueden ser:

1°.— Introducir eventuales sentencias de STOP hasta ahorquillar el problema. Con CONT podrá hacer seguir el programa con su rutina normal.

2°.— Introducir condicionantes del tipo IF-THEN STOP para analizar, por ejemplo, si las variables llegan con un valor adecuado a determinadas líneas de programa.

3°.— Si presume que algún bloque de su programa no funciona según lo previsto: ¡Anúlelo! Bien por borrado, bien con un salto incondicional del tipo GOTO.

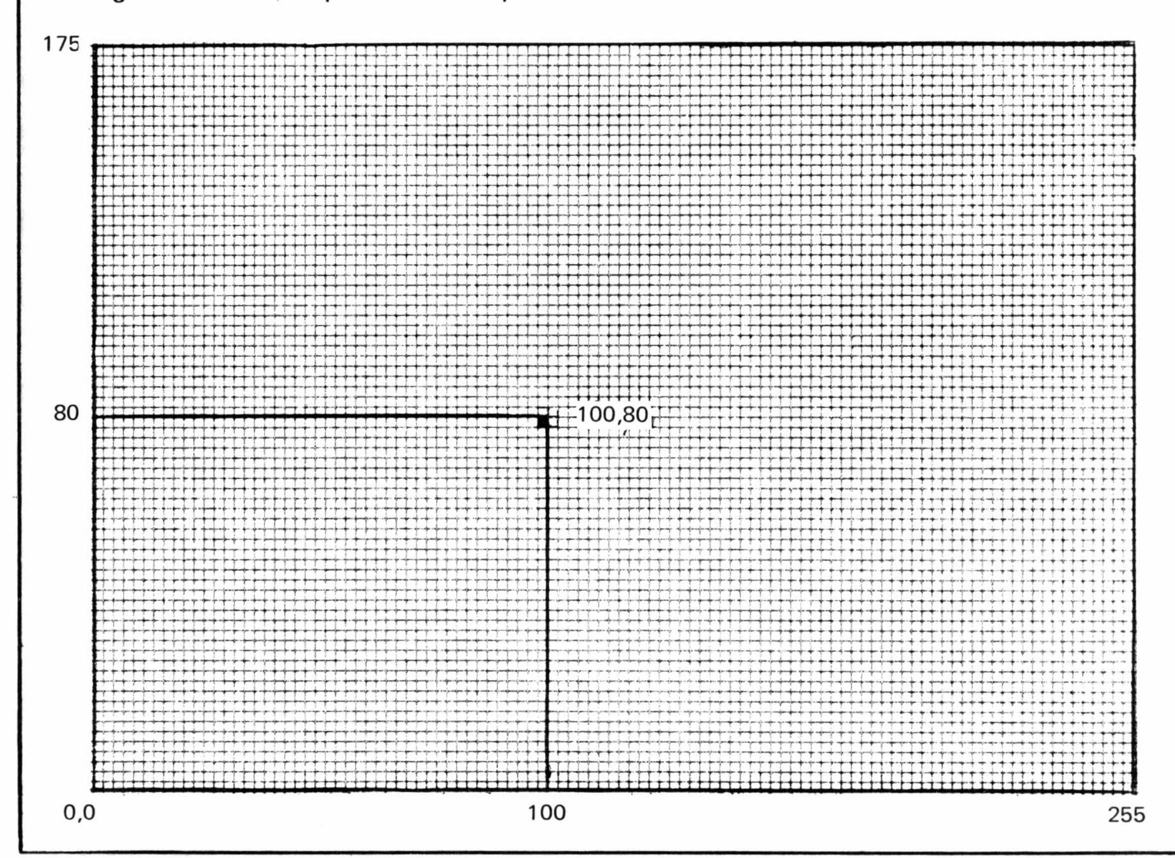
4°.— Introduzca ocasionales PRINT de variables para conocer los valores con que opera el programa en un sector, seguido, si necesita reflexionar sobre ellos, de un STOP.

Evidentemente, una vez localizado y subsanado el fallo, deberá eliminar todas las instrucciones suplementarias introducidas.

3-7	ALTA RESOLUCION DE GRAFICOS

Como ya se dijo anteriormente, trabajar en alta resolución de gráficos implica hacer trabajar el ordenador bastante más que cuando actúa y controla caracteres o, lo que es igual, manejar mallas completas de 8 x 8 cuadritos. Evidentemente, actuando sobre caracteres —el Spectrum— debe manejar 704 posiciones posibles de carácter, mientras que en alta resolución pasaría a controlar 45.086 cuadritos.

Como ya sabemos, con una instrucción PRINT AT (línea, columna) posicionaremos un grupo de 64 cuadritos (una malla de 8 x 8), mientras que, con una sentencia PLOT, situaremos exclusivamente un cuadrito. Estos cuadritos son la mínima porción de pantalla que el Spectrum puede controlar y que en inglés son conocidos como "**pixel**". Cualesquiera de estos cuadritos, estará situado en un plano coordenado cuyo eje de abcisas (horizontal) varía entre 0 y 255 y el de ordenadas (vertical) entre 0 y 175. Ambos ejes comienzan la numeración en el ángulo inferior izquierdo de la pantalla:



El cuadrito 100, 80 estará situado en la columna 100, fila 80. Obsérvese que PLOT, a diferencia de PRINT, considera primero la columna y después la fila y, muy importante, que la numeración comienza en el ángulo inferior izquierdo de la pantalla. Estas diferencias serán causa de no pocas dificultades cuando aparezcan ambas definiciones —alta y baja— en un mismo programa.

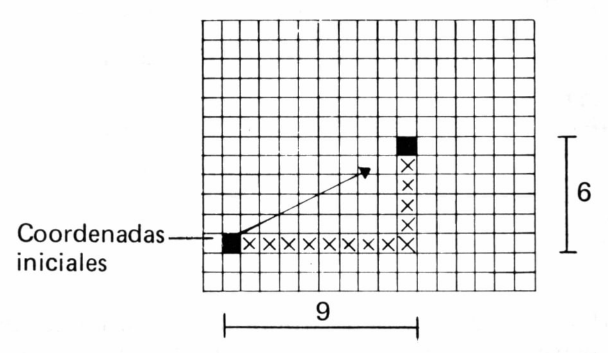
Estudiaremos a continuación los tres comandos del Spectrum para alta resolución de gráficos:

PLOT DRAW CIRCLE

Con **PLOT (x, y)** encenderemos —ponemos en color INK— el cuadrito situado en la columna **x**, fila **y**. Tanto x como y pueden ser números o cualquier expresión que finalmente den unos valores comprendidos entre 0 y 255, y 0 y 175 respectivamente.

DRAW x, y nos traza una recta desde el lugar donde estuviera situado el último cuadrito que hubiéramos manejado hasta otro cuadrito alejado del primero según los parámetros x, y que siguen a DRAW.

Es decir, si Vd. quiere trazar una recta desde una posición cualquiera hasta otra que esté, por ejemplo, 9 cuadritos horizontales a la derecha y 6 cuadritos verticales hacia arriba:



deberá teclear DRAW 9,6, bien entendido que si el cuadrito final está a la izquierda del inicial, deberemos colocar el signo - para indicar esta circunstancia. En el mismo orden de cosas habría que actuar para desplazar la recta en sentido vertical y hacia abajo.

Resumen de signos.

Dado un punto PLOT x, y inicial, las coordenadas del final de la recta deberán darse con signo positivo o negativo, según la dirección y el sentido de la recta.

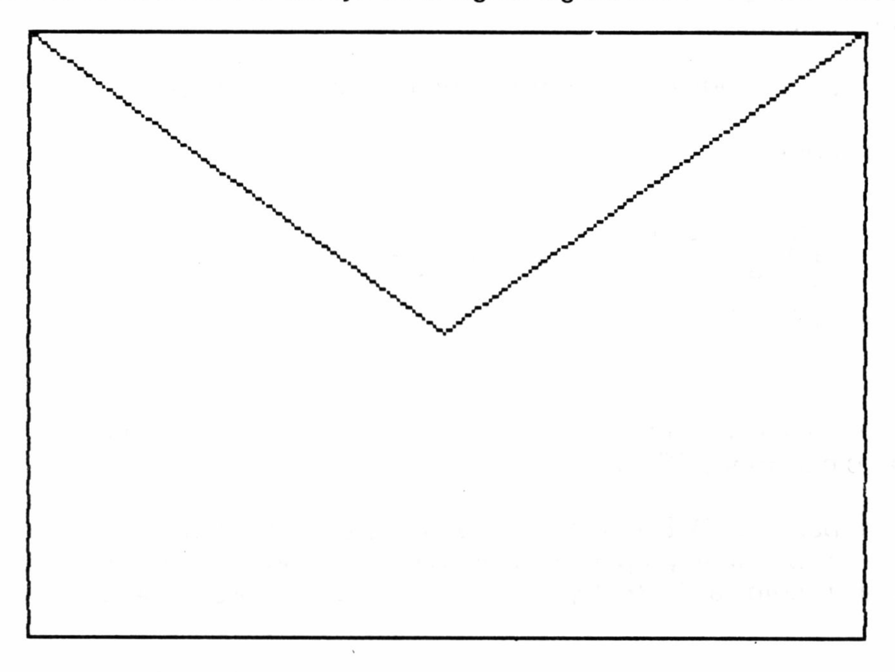
Recta	Coordenadas (final de la recta)	
	x	y
	+	+
	+	—
	—	—
	—	+
	Ø	+
	+	Ø
	Ø	—
	—	Ø

```
   10 INPUT "abscisa del cuadrito
   ";x
   20 INPUT "ordenada del cuadrit
o   ";y
   30 PLOT x,y
   40 INPUT "abscisa cuadrito fin
al   ";x1
   50 INPUT "ordenada cuadrito fi
nal   ";y1
   60 DRAW x1,y1
   70 GO TO 40
```

Con DRAW podemos trazar líneas quebradas, comenzando cada segmento donde acaba el anterior. Esta posibilidad se puede observar mediante el programa anterior:

Pruebe a iniciar el programa (líneas 10 y 20) con x = 0, y = 167. Continúe con x = 255, y = 0; después x = 0, y =-167. Siga con x = -255, y = 0 y finalice con x = 0, y = 167.

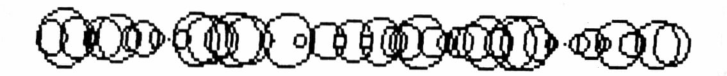
Para acabar este ejercicio, haga x = 127, y = -83 y después x = 127, y = 83. Y de esta forma se obtendrá el dibujo de la figura siguiente:



Para trazar una circunferencia con centro en el punto **x, y,** con radio **r**, bastará con utilizar **CIRCLE x, y, r.** Para comprobar esta función de forma inmediata apriete CIRCLE y a continuación 133, 88, 50. Ahora ENTER. Corra el programa.

```
 5 BORDER 1: INK 1: PAPER 6
10 FOR x=10 TO 245 STEP 5
20 LET r=RND*10
30 CIRCLE x,88,r
36 IF x>70 THEN GO TO 40
40 NEXT x
```

y obtendremos una sucesión de circunferencias de radios aleatorios, comprendidos entre 0 y 1, similar a ésta:



Tanto INVERSE como OVER pueden ser usados en alta resolución, ya que sólo actúan sobre un solo cuadrito y su forma de trabajar es bastante más simple que sobre caracteres.

INVERSE 1 puede considerarse como un sistema de borrado.

Por ejemplo:

```
   5 BORDER 1: INK 1: PAPER 6
  10 PLOT 0,0: DRAW 255,175
  20 FOR i=0 TO 175: NEXT i
  30 PLOT 0,0: DRAW INVERSE 1,25
5,175
```

Para conseguir el efecto de borrado se debe seguir la misma dirección de trazado pero con INVERSE 1.

Con respecto a OVER, si hay un puntillo en color INK en una determinada posición de pantalla y aplicamos un OVER 1 sobre esa posición, el puntillo cambiará su color a PAPER y al revés. El siguiente programa cambia algo con respecto al anterior. Pruebe y analice las variaciones.

```
   5 BORDER 1: INK 1: PAPER 6
  10 PLOT 0,0: DRAW 255,175
  20 FOR i=0 TO 175: NEXT i
  30 PLOT 0,0: DRAW OVER 1,255,1
75
  40 PAUSE 50
  50 PLOT 0,0: DRAW OVER 0,255,1
75
  60 GO TO 10
```

Si combinamos OVER 1 con INVERSE 1, ambos efectos se cancelan, ya que INVERSE 1 hace a todos los comandos de alta resolución donde se aplique, producir cuadritos en color PAPER. Según lo visto anteriormente, con OVER 1 un puntillo en color PAPER pasa a tomar el color INK. Por tanto nada cambió.

Una vez aquí, es conveniente resaltar un par de cosas. Primero, para trazar figuras en pantallas donde ya exista alguna impresión previa —la cual no se quiera hacer desaparecer ni tocar— es de uso OVER 1, tanto para su dibujo como para su borrado.

Por ejemplo:

```
  5 BORDER 1: INK 1: PAPER 6
 10 PRINT AT 10,5;"Los colores
del Spectrum"
 20 PLOT 0,0: DRAW OVER 1;255,1
75
 30 PAUSE 75
 40 PLOT 0,0: DRAW OVER 1;255,1
75
```

En segundo lugar, una instrucción del tipo PLOT INVERSE 1, OVER 1, x, y será una forma, a veces muy útil, de posicionar el cursor de los pixels "invisiblemente" donde nos interese.

POINT es otra función que actúa sobre cuadritos individuales y su misión es indicarnos si ese cuadrito está encendido o apagado (INK o PAPER), respondiendo 0 para apagado (color PAPER) ó 1 para encendido (color INK).

Ejemplo:

```
  5 BORDER 1: INK 1: PAPER 6
 10 PRINT AT 20,1;"█"
```

Al correr este programa habremos puesto en color INK una malla de 8 x 8 cuadritos, en la posición de carácter situada en la línea 20, columna 1, o lo que es igual, todos los cuadritos situados entre las abcisas 8 y 15, y entre las ordenadas 8 y 15 (Mire el desplegable de la página 17).

Si en estas condiciones probamos, por ejemplo, la sentencia directa PRINT POINT (12, 12), obtendremos un 1 (encendido) como respuesta del ordenador. Si por el contrario, y una vez corrido el programa nuevamente, tecleamos PRINT POINT (20, 20) la respuesta sería Ø, ya que el "pixel" elegido está fuera del carácter azul (INK 1) impreso por nuestro programa.

Puesto que los atributos controlan una posición de carácter completa (malla de 8 x 8 cuadritos), nunca afectarán a un sólo cuadrito individual, lo cual traerá algunas complicaciones que se deben conocer. Así, mientras podemos usar INK, PAPER, BRIGHT y FLASH dentro de sentencias de alta resolución, los atributos siempre actuarán sobre toda la malla de la posición de carácter donde los cuadritos controlados por PLOT, DRAW o CIRCLE estuvieran. Dicho de otra forma, dentro de una posición de carácter no pueden existir más de dos colores: el correspondiente a INK y el correspondiente a PAPER, y a ambos los controla la última instrucción INK/PAPER que haya recibido.

```
 10 CIRCLE INK 2,128,88,50
 20 PLOT 0,0: DRAW INK 5,255,17
5
```

En este programa, debemos observar cómo todas las posiciones de carácter donde van cayendo los sucesivos cuadritos encendidos de la circunferencia, van tomando color rojo. En los puntos donde se cortan la circunferencia y la recta, toda la posición de carácter afectada toma el color azul para INK (cuadritos encendidos de la malla).

Para afianzar la idea de la alta resolución vamos a desarrollar y comentar unos programas:

```
  10 INPUT "color del borde? ";b
  20 INPUT "color del papel? ";p
  30 INPUT "color de la tinta? "
;t
  40 INPUT "abscisa inicial?  ";
x
  50 INPUT "ordenada inicial?  "
;y
  55 BORDER b: INK t: PAPER p: C
LS : PAPER p
  60 PLOT OVER 1,x,y
  61 LET d$=INKEY$
  62 IF d$="d" THEN GO SUB 500
  70 GO SUB 400
  75 IF y<10 THEN PRINT AT 0,5;"
 "
  77 IF y<100 THEN PRINT AT 0,6;
" "
  80 PRINT AT 0,0;x;TAB 3;",";TA
B 4;y
 200 FOR f=0 TO 1
 210 PLOT OVER f;x,y
 220 NEXT f
 230 GO TO 60
 300 IF x<0 THEN LET x=255
 310 IF x>255 THEN LET x=0
 320 IF y<0 THEN LET y=175
 330 IF y>175 THEN LET y=0
 340 RETURN
 400 LET a$=INKEY$
 410 IF a$="5" THEN LET x=x-1
 420 IF a$="8" THEN LET x=x+1
 430 IF a$="6" THEN LET y=y-1
 440 IF a$="7" THEN LET y=y+1
 450 GO SUB 300
 460 RETURN
 500 INPUT "abscisa? ";x1
 510 INPUT "ordenada? ";y1
 520 DRAW x1,y1
 522 LET x=x+x1: LET y=y+y1
 530 RETURN
```

De la línea 200 a la 220 tenemos un bucle de parpadeo. de la 300 a la 330 tenemos las condiciones en los bordes.

Para conseguir una serie de "pisos" en la pantalla con "agujeros" que se desplacen por ellos, podemos estudiar el siguiente desarrollo:

```
  10 BORDER 5: INK 1: PAPER 6
  20 PLOT 0,50: DRAW 255,0
  30 PLOT 0,100: DRAW 255,0
  40 PLOT 0,150: DRAW 255,0
  50 PLOT 0,3: DRAW 255,0
  75 LET x1=0
  80 FOR x=0 TO 255
 150 PLOT OVER 1,x,100: IF x>15
THEN LET e=x-15: PLOT OVER 1,e,1
00
 180 PLOT OVER 1,x,50: IF x>15 T
HEN LET e=x-15: PLOT OVER 1,e,50
 230 PLOT OVER 1,x,3: IF x>15 TH
EN LET e=x-15: PLOT OVER 1,e,3
 240 IF x<50 THEN GO TO 270
 250 PLOT OVER 1,x1,3: IF x1>15
THEN LET w1=x1-15: PLOT OVER 1,w
1,3
 260 LET x1=x1+1: IF x1=255 THEN
 LET x1=0
 270 PLOT OVER 1,255-x,150: IF x
>15 THEN LET e=255-x-15: PLOT OV
ER 1,e,150
 290 PLOT OVER 1,x,150: IF x>15
THEN LET e=x-15: PLOT OVER 1,e,1
50
 320 NEXT x
 330 GO TO 80
```

Es conveniente acostumbrarse a tratar de interpretar los listados antes de recurrir a los comentarios. En cualquier caso el programa anterior nos define, en la primera línea, los colores con los que vamos a trabajar. A continuación posicionan los cuadritos de arranque y finales de un conjunto de rectas horizontales. Entre las líneas 80 y 320 se establece un bucle para conseguir, gracias a la instrucción OVER, la sensación de que los agujeros se desplazan sobre las líneas, obteniéndose pantallas de este tipo:

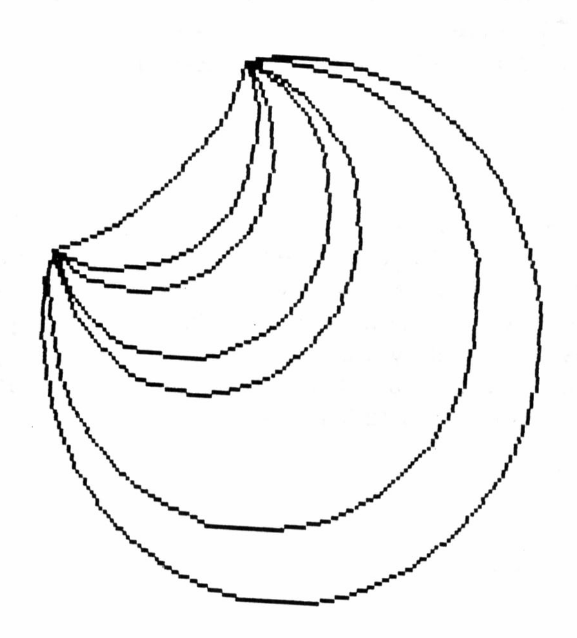
Con estos listados se pueden desarrollar juegos muy atractivos.

¿Se imagina a D. Pepe corriendo sobre estas líneas, tratando de saltar a través de los agujeros superiores y evitando caerse por los inferiores?

Para finalizar este epígrafe, diremos que el comando DRAW puede ser usado para trazar arcos de circunferencias siempre que éstos se midan en radianes.

Por ejemplo:

```
   5 INPUT "Pulse un valor de 0
a 5 ";m
  10 PLOT 100,100
  20 DRAW 50,50,m
  30 GO TO 5
```



Antes de poder percibir el significado de sentencias tales como PEEK y POKE, necesitaremos conocer algunos conceptos generales sobre los procesos internos en un computador.

Todo se basa en la electrónica digital. No se preocupe. Ni nos hace falta, ni tocaremos esta rama de la técnica, pero sí necesitamos conocer aquellos aspectos que afectan directamente a un programador.

Digital: Perteneciente o relativo a los dedos. Esta es la definición que nos da el diccionario.

¿Y qué tienen que ver los dedos con los ordenadores? pensará Vd. Pues bien, todo viene de antiguo.

Los seres humanos tenemos diez dedos, cinco en cada mano, lo cual ha hecho normal contar cualquier cantidad como el número de veces que contiene, esa cantidad, los diez dedos. Así 24 naranjas corresponden a 2 veces 10 dedos más cuatro dedos: 2 x 10 + 4 = 24.

24 es un número, y el 2 y el 4 también lo son, pero los números comprendidos entre el 0 y 9 —ambos inclusive— son conocidos como "digitos" para darnos a entender que son aquéllos susceptibles de ser contados directamente por los dedos de nuestras dos manos. Es decir son los símbolos fundamentales del sistema decimal que es, lógicamente, el de uso corriente entre las personas.

En otro orden de cosas, pero profundamente ligado a lo anteriormente expuesto, los ordenadores solo pueden reconocer si un impulso eléctrico puede —o no— pasar por un interruptor. Si pasa le asignamos un 1, en caso contrario un 0. Vamos, que un ordenador tiene dos dedos y, consiguientemente, sólo dos dígitos básicos —1 y 0— de un sistema de numeración de base dos o binario. Esto quiere decir que, al tener dos dígitos, sólo podrá interpretar números formados por esos dígitos o, lo que es lo mismo, series de ceros y unos.

Cualquier número en base diez tiene su equivalente en base dos y, lógicamente, al revés. Por ejemplo, 17 en base diez se representa por 10001 en base dos. Puede estudiar esto con más profundidad. Vea la página 217 del manual.

Conocido ya porqué los ordenadores utilizan el sistema de numeración binario (o base dos) vamos a ver cómo lo manejan; para ello necesitamos conocer el significado de algunas palabras.

3-8	PROCESOS INTERNOS DEL SPECTRUM. CONCEPTOS GENERALES

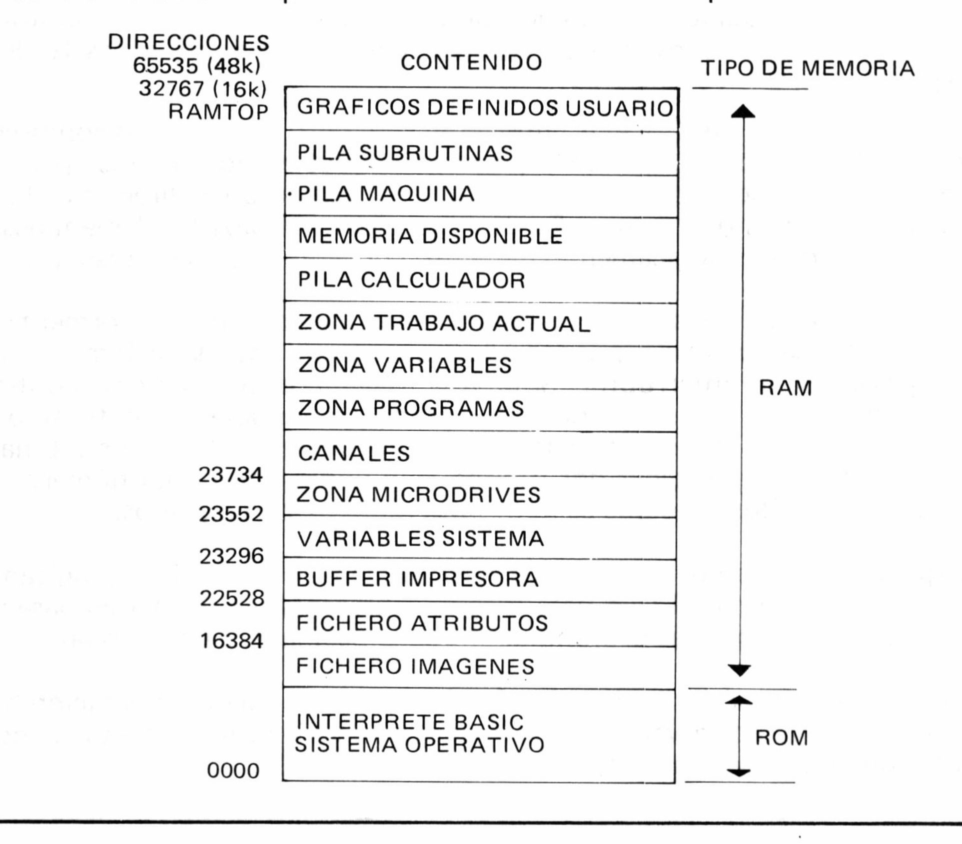
Un **bit** (se pronunca bit) nos indica uno de los dos estados —o dígitos— que un computador puede interpretar, y el número de bits que maneja dependerá de su microprocesador —verdadero cerebro del sistema— el cual, en el caso del Spectrum, es un Z80 de 8 bits que, consiguientemente, permite representar números comprendidos entre 0 y 255 —en base decimal— o, lo que es igual, puede hacer combinaciones de unos y ceros situados entre 00000000 y 11111111. A cualquiera de cada una de estas posibles combinaciones de 8 bits, se la conoce por un **byte** (se pronuncia bait) en su original vocablo anglosajón. O como un **octeto.**

Cada uno de estos bytes lo guarda el ordenador dentro de su memoria, en una posición definida, por lo que se conoce como una **dirección de memoria.**

La memoria está organizada como una secuencia lineal de direcciones o, sirva el ejemplo, como una calle con todas las casas en la misma acera, 8 pisos por casa y un sólo vecino por piso. Estos vecinos solo pueden ser Ø ó 1.

Los números binarios ubicados en sus respectivas direcciones de memoria, son interpretados por el computador como números, caracteres o instrucciones dependiendo de su situación en la memoria.

La memoria está organizada en áreas, las cuales ocupan espacios fijos o variables dentro de un orden preestablecido denominado mapa de memoria:



Como se puede ver, hay dos zonas claramente definidas, la inferior corresponde a la memoria ROM (Read-Only Memory) y a la cual sólo se puede acceder para "leer" el contenido de la misma y la memoria RAM (Random Acces Memory) que puede servir tanto para escribir como para leer información. En otras palabras, la memoria RAM puede ser usada para almacenar o reclamar información, mientras que la ROM sólo sirve para reclamar aquella información que el fabricante haya introducido en ella y contiene, en el caso del Spectrum y entre otras cosas, el "interpreter"—o traductor simultáneo— del lenguaje BASIC.

La capacidad de memoria de un ordenador es función de la cantidad de información que pueda almacenar pero, siendo la unidad elemental de memoria el byte, tendremos que la memoria se mide en bytes o, más usualmente, en kilo-bytes, Kb o K sin más. Una K hace referencia a mil bytes, pero es simplemente una aproximación nemotécnica, ya que una K contiene exactamente 1024 bytes.

Así, un Spectrum de 16 K RAM indica que su capacidad de almacenamiento es de 16384 (16 x 1.024).

Dicho esto, podemos entender ya que cualquier posición de memoria puede guardar un número comprendido entre 0 y 255. Ahora veremos que un carácter se almacena como un conjunto de 8 bits.

En la página 183 del manual del Spectrum está el juego de caracteres. La primera columna es la denominada "código" (CODE) y está formada por números que van del 0 al 255 y la segunda es la denominada "carácter" (CHR). Esto significa que el código 65 está asimilado al carácter 65, de tal forma que si tecleamos PRINT CHR$ (65) obtendremos la letra A por respuesta. En definitiva cualquier posición de memoria puede contener un número entre 0 y 255, el cual será interpretado de acuerdo con la segunda columna de la tabla de caracteres del Spectrum.

Estas instrucciones están dirigidas a darnos información o manipular sobre las posiciones de memoria.

Pero, quizá, lo primero a saber es qué interpretación tendría —en español— estas dos palabras inglesas:

PEEK - Atisbar **POKE** - Hurgar

Bien, pues realmente esto es lo que nos permiten hacer. Con PEEK atisbamos lo que hay en una determinada posición de memoria definida por su dirección. Así, por ejemplo, si pretendemos conocer el contenido de la dirección de memoria 18121 bastaría con teclear PRINT PEEK 18121. Dado que PEEK siempre nos devolverá un número decimal —sin que esto implique que no pueda representar un número, un carácter o una instrucción— lo podríamos asimilar, si fuera de interés, a una variable numérica: LET a = PEEK 18121.

Quede claro que PEEK sólo "atisba" y, por tanto, no modifica ni altera nada de lo ya existente en memoria. Dicho con claridad: "no es peligrosa".

Con POKE la cosa cambia. Ya no se atisba, se hurga. Nos dirigimos a una dirección de memoria no para "atisbar" sino para "insertar" un determinado valor en esa posición y, por tanto es "peligrosa" ya que podríamos modificar posiciones situadas dentro de sectores de memoria vitales (memoria ROM).

Con POKE almacenamos un byte en **cualquier** posición de memoria, incluso en una que ya está habilitada por Vd., con lo cual perdería su contenido en favor del nuevo.

Como ejemplo vamos a teclear POKE 18121,65 almacenando, de esta forma, en la dirección 18121 el número decimal 65, con lo cual un PRINT CHR$ (PEEK 18121) nos da una A en pantalla.

Cabría pensar que un programador sólo necesita conocer la sintaxis del lenguaje de programación que utiliza y todas las palabras del mismo pero, es el caso que, a medida que domina las interioridades de su computador, puede obtener soluciones que serían inalcanzables sin tales conocimientos. Por tanto rogamos al lector un esfuerzo más, garantizándole que quedará sobradamente compensado.

Antes de ir más allá, recordaremos que el número más grande que se puede almacenar en una posición de memoria cualquiera está compuesto por 8 bits —que equivale a un byte— que corresponde a una combinación de ocho "ceros" y "unos". Por tanto, la combinación mínima vendrá dada por 00000000 y la más alta por 11111111, esta representación pertenece al sistema binario. Si lo expresáramos en el sistema decimal diríamos que en una posición de memoria se puede almacenar un número entre 0 y 255.

Así, si usamos una posición de memoria para contar, empezaremos por 0 y seguiremos sin dificultad hasta 255 pero, si aquí intentamos añadir otra unidad, no almacenaremos el número 256, sino que comenzaríamos otra vez a contar desde el cero. Pues bien, para alcanzar números superiores a 255, el computador habilita otra posición de memoria de tal forma que, cada vez que la anterior posición comienza a 0, la nueva aumenta una unidad. En estas condiciones, podríamos seguir contando en la primera posición de memoria como si nada hubiera pasado, mientras la segunda posición de memoria actúa como un contador del número de veces que la primera ha alcanzado 255.

La primera ubicación cuenta números entre 0 y 255 y es conocida como el **byte menos significativo** (LSB. Least Significant Byte) y la segunda ubicación cuenta de 256 en 256, y es conocida como el **byte más significativo** (MSB. Most Significant Byte).

Con un PEEK sobre el LSB, obtendremos un número entre 0 y 255 que representa justamente ese número, pero un PEEK sobre el MSB nos dará el número de veces que contiene 256. Esto, en términos que entienda el BASIC y siendo **n** la posición de memoria del LSB, sería:

PEEK n + 256 * PEEK (n + 1)

obteniendo de esta forma el número que está almacenado en estas dos posiciones de memoria, el cual, evidentemente, irá desde 0 hasta 65.535.

En sentido contrario podemos trocear un número para almacenarlo en dos ubicaciones de memoria; para ello lo dividiremos previamente entre 256 para saber el número de veces que lo contiene y almacenarlo en el byte más significativo (MSB) y entonces almacenar el resto en el byte menos significativo (LSB).

Esto es:

POKE n + 1, INT (N/256)
POKE n, N - 256 * INT (N/256)

Siendo **n** la dirección del LSB y **N** el número a trocear. Con POKE lo que hacemos es colocar los trozos adecuados en las direcciones de memoria seleccionadas.

Una vez en este punto echemos un vistazo a la página 174 del manual del Spectrum, segunda de las dedicadas a **las variables del Sistema**. En la primera línea se nos indica que las direcciones 23606 y 23607 están dedicadas a conocer las ubicaciones de memoria del juego de caracteres del Spectrum, de tal forma que, para obtener la dirección de memoria donde se inicia la tabla de caracteres, deberemos hacer la siguiente operación —de acuerdo con lo visto anteriormente—:

PEEK 23606 + 256 * PEEK 23607 + 256 = 15616

Los dos primeros sumandos representan un número decimal comprendido entre 0 y 65535 el cual, y según se indica en el manual, nos da la dirección de arranque (menos 256) del conjunto de caracteres.

El número de la dirección de arranque obtenido, sitúa el conjunto de caracteres dentro de la zona de memoria ROM que va desde la dirección 0 a la 16383, lo cual es lógico si tenemos en cuenta que el computador, una vez conectado, está en condiciones de trabajar gracias a su memoria ROM donde guarda toda la información básica del sistema y, entre ella, el juego de caracteres.

Este juego de caracteres, según el apéndice A del manual, va desde el **SPACE** (código 32) hasta el símbolo © (código 127).

Por otra parte, ya vimos que todo carácter está contenido en una malla de 8 x 8 cuadritos y que cada fila de esta malla es una combinación de "unos" y "ceros", para diferenciar los cuadritos encendidos de los apagados.

Por ejemplo:

0 0 0 0 0 0 0 0	....	1ª fila	0
0 0 1 1 1 1 0 0	....	2ª fila	60
0 1 0 0 0 0 1 0	....	3ª fila	66
0 1 0 0 0 0 1 0	....	4ª fila	66
0 1 1 1 1 1 1 0	....	5ª fila	126
0 1 0 0 0 0 1 0	....	6ª fila	66
0 1 0 0 0 0 1 0	....	7ª fila	66
0 0 0 0 0 0 0 0	....	8ª fila	0

En el ejemplo anterior hemos representado el modelo de malla de 8 x 8 de cuadritos encendidos y apagados que el ordenador tiene en su memoria ROM para la letra A y, dado que una dirección de memoria sólo puede guardar un byte, deducimos que cada carácter necesita 8 direcciones de memoria, una para cada fila, y de esta forma ser "memorizado".

La columna de la derecha son los números decimales que corresponde a los números binarios que representan los "ceros" y "unos" de cada fila de la malla de 8 x 8 y, en definitiva, serían los números decimales obtenidos al aplicar PEEK sobre cada una de las 8 direcciones de memoria en que está almacenado el carácter.

La cuestión ahora vendría al querer saber dónde está —o cuál es— la dirección de memoria de la primera fila de "ceros" y "unos" de cualquier carácter. Para responder a esto debemos recordar que la dirección de memoria donde se inicia la tabla de caracteres es: PEEK 23606 + 256 * PEEK 23607 + 256 = **da (dirección de arranque)** y también que la tabla comienza en el código 32 (SPACE). Por lo que de ambas cosas es fácil deducir que la dirección de memoria inicial de un carácter cualquiera —**c**— viene dada por:

dc = da + 8* (c-32)

Siendo **da** la dirección de arranque ya citada y **dc** y **c** la dirección y el código del carácter en cuestión.

Quiere decirse, que un **PEEK dc** nos daría el contenido, en decimal, de esa posición de memoria. Con una simple transformación a binario habríamos obtenido la combinación de "ceros" y "unos" que la memoria ROM guarda como modelo de la primera fila del carácter c.

En este momento ya conocemos **dónde** y **cómo** guarda en memoria el ordenador el conjunto de caracteres.

Una aplicación típica de todo lo tratado en este capítulo es el trazado de caracteres y cadenas de caracteres de gran tamaño.

Para llegar a un programa que nos haga tal trabajo, iremos por partes. Con la rutina que sigue, imprimiremos en pantalla la fila de arranque, en "ceros" y "unos" del carácter elegido como modelo para esta primera etapa.
(ver tabla de conversión).

```
  10 INPUT "seleccione caracter
";c$
  20 IF LEN c$>1 THEN GO TO 10
  30 LET da=PEEK 23606+256*PEEK
23607+256
  40 LET dc=da+8*(CODE c$-32)
  50 LET f=PEEK dc
  60 FOR i=0 TO 7
  70 LET b=f-2*INT (f/2): LET f=
INT (f/2)
  80 PRINT AT 21,(20-2*i);b
  90 NEXT i
```

Para conseguir el carácter en cuestión completo, en su malla de "ceros" y "unos", sólo tendríamos que añadir un bucle FOR/NEXT:

```
  10 INPUT "seleccione caracter
";c$
  20 IF LEN c$>1 THEN GO TO 10
  30 LET da=PEEK 23606+256*PEEK
23607+256
  40 LET dc=da+8*(CODE c$-32)
  45 FOR j=0 TO 7
  50 LET f=PEEK (dc+j)
  60 FOR i=0 TO 7
  70 LET b=f-2*INT (f/2): LET f=
INT (f/2)
  80 PRINT AT 21,(20-2*i);b
  90 NEXT i
 100 PRINT ''
 110 NEXT j
```

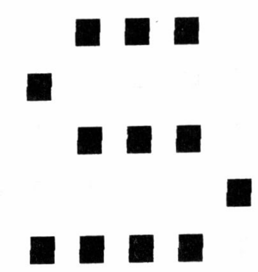
Al conocer este programa seleccione el carácter "s" y, si respondemos con una "y" al Scroll?, veremos el conjunto de 0 y 1 que forman la malla de 8 x 8 del carácter elegido, como se puede apreciar en la representación que sigue:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 1 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 1 0 0
0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
```

Si allá donde aparezca un 0 dejamos su posición de carácter en color papel (PAPER) y donde aparezca un 1 colocamos una posición de carácter en el color (INK) que deseemos, habremos obtenido una representación a escala del carácter introducido en c$, de esta forma:

```
  5 PAPER 6: INK 1: BORDER 1
 10 INPUT "seleccione caracter
";c$
 20 IF LEN c$>1 THEN GO TO 10
 30 LET da=PEEK 23606+256*PEEK
23607+256
 40 LET dc=da+8*(CODE c$-32)
 45 FOR j=0 TO 7
 50 LET f=PEEK (dc+j)
 60 FOR i=0 TO 7
 70 LET b=f-2*INT (f/2): LET f=
INT (f/2)
 75 IF b=0 THEN GO TO 90
 80 PRINT AT 21,(20-2*i);"█"
 90 NEXT i
100 PRINT ''
110 NEXT j
```

Corriendo este programa obtendremos una impresión de este tipo:



Pasar de la impresión de un solo carácter a una cadena de caracteres, se limita a añadir un bucle más para poder abarcar todos los caracteres de esa cadena:

```
  5 PAPER 6: INK 1: BORDER 1
 10 INPUT "entre texto  ";c$
 20 LET da=PEEK 23606+256*PEEK
23607+256
 30 FOR k=1 TO LEN c$
 40 LET dc=da+8*(CODE (c$(k TO
k))-32)
 45 FOR j=0 TO 7
 50 LET f=PEEK (dc+j)
 60 FOR i=0 TO 7
 70 LET b=f-2*INT (f/2): LET f=
INT (f/2)
 75 IF b=0 THEN GO TO 90
 80 PRINT AT 21,(20-2*i);"■"
 90 NEXT i
100 PRINT ''
110 NEXT j
120 NEXT k
```

En el mapa de memoria representado en el capítulo **"Procesos internos del Spectrum"** nos encontramos con el **"fichero de representación visual"** (Display file) que va desde la ubicación de memoria con dirección 16384 a la 22527, en cada una de estas posiciones de memoria se almacenará, desde que se conecta la máquina, una combinación de 8 "ceros" y "unos" —un byte— gracias a las cuales va a saber el ordenador qué cuadritos están apagados (0) o encendidos (1), en la zona controlada por INK y PAPER.

Como ya dijimos al principio, el ordenador considera dividida la pantalla del televisor en una malla de 256 cuadritos horizontales por 176 verticales, los cuales están apagados, inicialmente o, lo que es igual, todas las posiciones de memoria situadas entre 16384 y 22527 estarán ocupadas por 0.

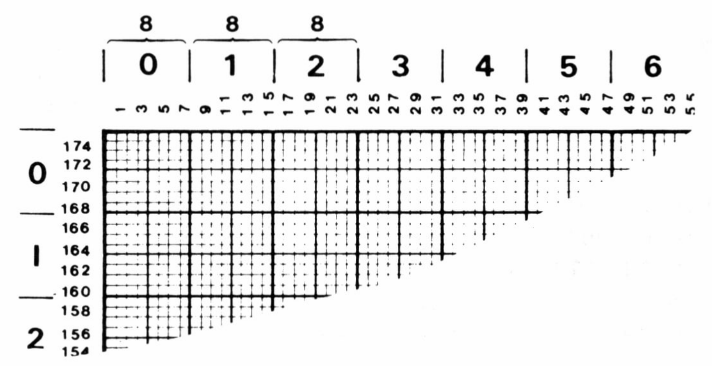
Pruebe:

```
   10 INPUT "direccion a chequear
?  ";d
   15 CLS
   20 IF d<16384 OR d>22527 THEN
GO TO 10
   30 PRINT PEEK d
```

Ahora bien, a medida que se van produciendo impresiones en pantalla, el ordenador debe ir memorizando diferentes combinaciones de 0 y 1 en el "Display file". La cuestión, que podría ser de interés según el tipo de aplicación a desarrollar, sería **cómo** se produce el almacenamiento y **cómo encontrar** la dirección de memoria de un puntillo determinado de pantalla.

Empezamos por el **"cómo".**

A cada dirección de memoria del "Display file" corresponden 8 cuadritos de pantalla y sólo esos ocho cuadritos. El criterio de correspondencia es el siguiente:



La primera dirección almacenará la situación encendido/apagado (1,0) de los ocho primeros cuadritos de la primera fila de la malla de 256 x 176, los cuales obviamente corresponden a la primera fila de cuadritos de la malla de 8 x 8 de la primera posición de carácter. La siguiente dirección almacenará los ocho cuadritos siguientes de la misma fila y así sucesivamente hasta el final de la misma. En este momento tendremos 32 direcciones de memoria ocupadas que corresponden a las primeras 32 posiciones de carácter de la primera línea de caracteres.

A continuación se almacenan los ocho cuadritos primeros de la novena línea de cuadritos de la malla de 256 x 176 que corresponden a la primera fila de cuadritos de la malla de 8 x 8 de la primera posición de carácter de la segunda línea de caracteres. La siguiente dirección guardará los ocho cuadritos siguientes de la misma fila (la novena) y así sucesivamente hasta el final de la misma:

Después, y de la misma forma, los cuadritos de la fila 17 (tercera línea de caracteres) y así hasta llegar a la primera línea de cuadritos de la octava línea de caracteres.

En este punto se repite el proceso con todas las segundas filas hasta llegar a la octava línea de posiciones de carácter y así, hasta llegar a la octava fila de cuadritos de la octava línea de posiciones de carácter.

En este momento el ciclo se repite completo entre la primera fila de cuadritos de la novena línea de posiciones de carácter y la octava fila de cuadritos de la décimo sexta línea de posiciones de carácter.

Y el ciclo se repite entre la primera fila de cuadritos de la décimo séptima línea de posiciones de carácter y la octava fila de cuadritos de la vigésimo cuarta línea de posiciones de carácter (incluyendo, claro está, las dos líneas reservadas a entrada de datos y emisión de mensajes).

Para ver gráficamente todo lo anteriormente expuesto, introduzcamos en todas las direcciones de memoria del "Display file" una combinación de ocho "unos" (11111111); este número binario corresponde al 255 decimal, que será el código que usemos para hurgar (POKE) en las direcciones que nos interesen; una vez hecho esto procederemos a su impresión:

```
10 FOR d=16384 TO 22527
20 POKE d,255
30 NEXT d
40 FOR i=16384 TO 22527
50 PRINT AT 0,0;PEEK i
60 PAUSE 50
65 PRINT AT 0,0;"    "
67 PAUSE 20
70 NEXT i
```

Líneas 10-30: Con estas instrucciones, modificamos de apagado (0) a encendido (1), todas las direcciones del "Display file".

Líneas 40-70: Con estas instrucciones obtendremos los números decimales, correspondientes a los binarios representativos de la situación en que se encuentran cada dirección del "Display file".

Corra este programa.

Observe los dos ceros que aparecen en segundo y tercer lugar. Si Vd. ha comprendido el párrafo anterior, sabrá a qué son debidos.

Hasta aquí se ha visto **cómo** se produce la memorización del contenido de la pantalla. Ahora veremos como, dada una posición en pantalla de un grupo de 8 cuadritos determinados, localizar su dirección de memoria en el "Display file".

En la terminología aplicada en la explicación previa referente al "cómo", se han utilizado términos tales como **"Línea de posiciones de carácter"**, para hacer referencia a la posición vertical de las mallas de 8 x 8 donde los caracteres se producen, y **"filas de cuadritos"** para especificar la situación relativa de los cuadritos dentro de las Líneas de Posición de carácter.

Utilizando estas expresiones diríamos que un grupo de 8 cuadritos está, por ejemplo, en la **línea de posición de carácter cuarta**, y en su tercera posición —de izquierda a derecha—, ocupando la séptima fila de cuadritos dentro de esa posición de carácter, de tal forma que si nosotros hacemos:

Línea de posición de carácter = **L** (varía entre 0 y 23)

Carácter de la línea = **C** (varía entre 0 y 31)

Fila de cuadritos = **F** (varía entre 0 y 7)

Cualquier grupo de 8 cuadritos dentro de la pantalla quedará posicionado con **L, C** y **F**.

3-10	MAS SOBRE LA PANTALLA

Para llegar a una fórmula que nos dé la dirección de memoria de cualquiera de estos posibles grupos de 8 cuadritos, debemos partir de la primera dirección (16384) dedicada al control de pantalla. Después debemos averiguar el número de grupos completos de 8 líneas de carácter, que separan a nuestro grupo de 8 cuadritos del origen de la pantalla, cuyo valor puede ser uno de estos:

Caracteres en una línea		Líneas de posición de carácter		Filas carácter	
		0	x 8	= 0	
		1			
		2			
32	x	3			El grupo de 8 cuadritos está entre líneas 0 y 7
		4			
		5			
		6			
		7			
		8	x 8	=2048	
		9			
		10			
32	x	11			El grupo de 8 cuadritos está entre líneas 8 y 15
		12			
		13			
		14	x 8	=4096	
		15			
		16			
		17			
		18			
32	x	19	x 8	=6144	Entre líneas 16 y 23
		20			
		21			
		22			
		23			
					La última posición posible es 6143

Esto, en términos matemáticos y BASIC, es igual a **2048* INT (L/8),** siendo **L** lo ya indicado. Una vez que sabemos en cuál de los tres sectores horizontales de la pantalla está situado el "byte" que nos interesa, deberemos determinar cuántas líneas completas (a 32 caracteres cada una) existen entre el comienzo del sector en cuestión y la posición de carácter donde se encuentra situado nuestro grupo de 8 cuadritos:

32 * (L - 8* INT (L/8)).

Ahora determinaremos el número de filas de cuadritos completas (a razón de 256 cuadritos cada una) que no han llegado a completar una línea de posición de carácter: 256*F. Por último, la fila de cuadritos incompleta que viene representada directamente por el valor de C.

Recopilando todos estos componentes llegamos a:

d = **16384 + 2048* INT (L/8) + 32* (L - 8 * INT (L/8))**

Un modo de comprobar esta fórmula es dar valores a las variables **(L, C y F)** para unos valores conocidos de **d**. Este sería el caso, p. e., de la primera y última posiciones posibles de pantalla, las cuales tienen por direcciones de memoria d = 16384 y d = 22527.

```
  10 INPUT "linea de posicion de
 caracter? ";l
  20 INPUT "caracter en la linea
? ";c
  30 INPUT "fila de cuadritos? "
;f
  40 LET d=16384+2048*INT (l/8)+
32*(l-8*INT (l/8))+256*f+c
  50 PRINT "Direccion de memoria
 ";d
  60 GO TO 10
```

Pruebe estas dos series:

1°.— L = 0, C = 0, F = 0 . . . . d = 16384

2°.— L = 23, C = 31, F = 7 . . . . . d = 22527

Volviendo de nuevo al mapa de memoria podemos ver que, entre las direcciones 22528 y 23295, se encuentra situado el **fichero de atributos** (Attributes file) el cual almacena la información sobre el color (INK y PAPER), brillo (BRIGHT) y parpadeo (FLASH) de cada posición de carácter sobre un total de 24 x 32 posiciones de carácter.

Hay 768 direcciones de memoria disponibles en el "fichero de atributos" y un total de 768 posiciones de carácter en pantalla, lo cual implica que los atributos de cada posición de carácter son guardados y controlados por un solo byte o, como ya sabemos, por una combinación de ocho "ceros" y "unos".

3-10	MAS SOBRE LA PANTALLA

Siguiendo el sistema aplicado con el "Display file", empezaremos por conocer **cómo** se produce el almacenamiento y a continuación analizaremos **cómo encontrar** la dirección de memoria de un carácter cualquiera. Más un tercer factor. De qué forma un byte controla los cuatro atributos: INK, PAPER, BRIGHT y FLASH.

El primer interrogante tiene una respuesta sencilla. El primer carácter de la primera línea de posición de carácter corresponde a la primera dirección del fichero de atributos, el segundo carácter de la misma fila a la segunda dirección de memoria y así hasta llegar al final —trigésimo segundo carácter— una vez en este punto, la siguiente dirección la ocupará, sencillamente, el primer carácter de la segunda línea de posiciones de carácter y, así, hasta el final de esta línea. El proceso se repite hasta la línea más baja de la pantalla.

Si mantenemos el significado de L, C y d dado anteriormente veremos que la dirección de memoria que ocupa el byte controlador de los atributos de un carácter cualquiera viene dado por:

$$\mathbf{d = 22528 + 32 * L + C}$$

Fórmula evidente.

Respecto a la forma en que un solo byte controla los atributos, está en función de las posibilidades que ofrece cada atributo. Así, FLASH requiere **un bit** ya que sólo puede estar activado (0) o desactivado (1), BRIGHT **un bit** por la misma razón. INK y PAPER necesitan **tres bits** cada uno ya que son ocho los colores a controlar por cada comando. Por tanto, y de acuerdo con el manual (pág. 219), tendremos:

* Primer bit -------- indica situación de FLASH.
* Segundo bit ----- indica situación de BRIGHT.
* Tercer bit
* Cuarto bit ------ indica color de PAPER.
* Quinto bit
* Sexto bit
* Séptimo bit --- indica color de INK.
* Octavo bit

Con estos conocimientos podríamos deducir la situación de los atributos a través de un PEEK d, pero, gracias a la función ATTR, nos evitamos investigar la dirección de memoria —d— correspondiente a un carácter cualquiera situado en la línea L y en la posición C de esa línea. Para ello bastará aplicar **ATTR (L,C)** y obtendremos un número decimal comprendido entre 0 y 255, ya que corresponderá a una combinación binaria entre 00000000 y 11111111.

La rutina que sigue nos dará la situación de los atributos mediante la adecuada transformación del número decimal obtenido por ATTR (L, C):

```
  10 INPUT "linea del caracter?
";L
  20 INPUT "posicion del caracte
r en la linea? ";C
  30 LET a=ATTR (L,C)
  40 LET flash=INT (A/128): IF f
lash=0 THEN PRINT "FLASH DESACTI
VADO": GO TO 60
  50 PRINT "flash activado"
  60 LET bright=INT ((a-flash*12
8)/64): IF bright=0 THEN PRINT "
BRIGHT DESACTIVADO": GO TO 80
  70 PRINT "bright activado"
  80 LET paper=INT ((a-flash*128
-bright*64)/8)
  90 PRINT "paper ";paper
 100 LET ink=INT (a-flash*128-br
ight*64-paper*8)
 110 PRINT "INK ";ink
 120 GO TO 10
```

```
FLASH DESACTIVADO
BRIGHT DESACTIVADO
paper 7
INK 0
```

Para comprobar este programa una vez tecleado, introduzca PRINT INK 6; AT 10, 10; "■" y, después córralo con un GOTO 10/Enter.

Pulse STOP y ENTER. Pruebe ahora PRINT INK 3; BRIGHT 1; FLASH 1; PAPER 6; AT 10, 10; "■". Observe las diferencias entre ambos.

3-11 CONSIDERACIONES FINALES SOBRE EL MAPA DE MEMORIA

Como se puede observar en el mapa de memoria, a continuación del "fichero de atributos" viene el control del **"buffer" de la impresora y las variables del sistema**, que serán explicadas a continuación, y el resto de las zonas de memoria que no son objeto de este libro.

El "buffer" es como un depósito de donde sale su contenido sólo cuando llega al nivel adecuado. En este orden de cosas el **buffer de la impresora** almacena información hasta que ha completado una línea de 32 posiciones de carácter. Esto implica que necesita 32 x 8 bytes, que son en efecto, las direcciones disponibles en esta zona (de 23296 a 23551).

Las **variables del sistema** abarcan un conjunto de direcciones de memoria que van desde la 23552 a la 23734. Estas variables pueden necesitar uno o dos bytes. Como ya sabemos, un byte sólo puede contener números entre 0 y 255 y dos bytes pueden llegar de 0 a 65535.

Las variables del sistema, en función de sus respectivos valores mantienen la organización del sistema. Conocer estos valores o modificar su contenido puede ser muy útil.

El conjunto de estas variables se puede ver en la página 173 del manual del Spectrum. Conocer un poco más sobre algunas de ellas, servirá de ejemplo para profundizar en el estudio del resto.

Dirección 23560

Un PEEK sobre esta dirección nos da el código de la última tecla apretada.

Ejemplo:

Si Vd. teclea directamente PRINT PEEK 23560 seguido de ENTER, observará en el ángulo superior izquierdo cómo aparece el número 13, que es el código de la tecla ENTER (pág. 183 del manual), lo cual es lógico ya que esa tecla fue la última utilizada.

Con el fin de observar cómo una adecuada combinación de las posibilidades que ofrece la máquina nos aumenta el campo de alternativas de uso del sistema, pruebe:

PRINT CODE INKEY$

seguido, claro está, de la tecla imperativa ENTER.

·Igualmente obtendremos 13.

Pruebe el siguiente programa y convertirá su ordenador en una rudimentaria máquina de escribir:

```
 10 PAUSE 25: IF INKEY$="" THEN
GO TO 10
 20 PRINT CHR$ PEEK 23560;
 30 GO TO 10
```

Dirección 23561

Un PEEK sobre esta dirección nos da 35. Pruebe:

PRINT PEEK 23561

En la pantalla aparecerá 35, que es el tiempo medido en 1/50 segundo, que debe estar apretada una tecla para iniciar su mecanismo de repetición automática.

Pruebe un POKE 23561, 0, seguido de ENTER por supuesto, y comprobará como la autorrepetición entra en funcionamiento mucho más tarde.

Un POKE 23561, 1, hace casi inmanejable el teclado.

Dirección 23562

PRINT PEEK 23562 nos dará 5, que es el tiempo medido en 1/50 segundo, transcurrido entre dos repeticiones de una tecla en su proceso de autorrepetición.

Si prueba el siguiente programa obtendrá el tiempo, en cincuentavos de segundos, que mantiene apretada una tecla cualquiera:

```
     5 LET a=0
     6 CLS : LET b=a
     7 IF INKEY$="" THEN PRINT AT
0,0;b: LET a=0: GO TO 7
    20 LET a=a+PEEK 23562
    30 GO TO 6
```

Si corre el programa y aprieta una tecla, verá aparecer en el ángulo superior izquierdo el número correspondiente al tiempo de la depresión.

El uso de esta dirección de memoria puede ser útil, entre otras cosas, para hacer reaccionar un gráfico cualquiera, con más o menos velocidad en función del tiempo que hemos mantenido apretada una determinada tecla.

Dirección 23606 (y 23607)

Estas direcciones y su utilidad fueron explicadas en el capítulo **Más sobre la memoria.**

Dirección 23609

Si tecleamos un PRINT PEEK 23609, ENTER, obtendremos Ø que es el tiempo que dura el ruido que simula la depresión de una tecla.

Si hacemos un POKE 23609, 10 queda un "click" muy audible.

Dirección 23625

Si en el transcurso de la ejecución de un programa cualquiera usted teclea:

PRINT PEEK 23625

Obtendrá el número de línea de programa donde está situado el cursor.

Dirección 23677

Un PRINT PEEK 23677 nos dará la abcisa del último cuadrito (pixel) posicionado en la pantalla.

Dirección 23678

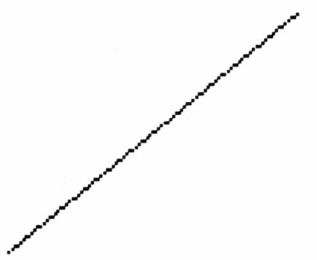
Un PRINT PEEK 23678 nos dará la ordenada del último cuadrito (pixel) posicionado en la pantalla.

Un ejemplo aclaratorio de uso para las dos direcciones anteriores, podría ser:

```
10 PLOT 2,7
20 DRAW 100,75
30 PRINT PEEK 23677,PEEK 23678
```

Con lo cual obtendríamos una pantalla similar a ésta:

```
102                  82
```



donde el número situado a la izquierda nos da el valor **x** del pixel donde acaba la recta y el número a la derecha nos da el valor de **y**.

Si una vez que se ha introducido un programa, se teclea —o introduce a modo de instrucción— la siguiente sentencia:

PRINT ((PEEK 23627 + 256 * PEEK 23628) – (PEEK 23635 +256 * PEEK 23636))

3-11	CONSIDERACIONES FINALES SOBRE EL MAPA DE MEMORIA

Obtendremos el número de "bytes" que ocupa un programa.

En el caso de que Vd. desee ampliar la memoria disponible, a costa de perder la posibilidad de crear sus propios gráficos, teclee lo que sigue:

Para un Spectrum 48 K	**Para un Spectrum 16 k**
POKE 23675, 255	**POKE 23675, 255**
POKE 23676, 255	**POKE 23676, 255**
CLEAR 65534	**CLEAR 32766**

Dirección 23688

Si Vd. teclea esta instrucción:

PRINT AT 1Ø, 1Ø; "p";

y a continuación:

PRINT PEEK 23688

nos aparecerá el número 22, justo a continuación de la letra **p**, que corresponde a la diferencia existente entre 33 y la columna donde debe ser impreso —sobre la pantalla— el siguiente carácter.

Dirección 23689

Si repetimos: PRINT AT 10, 10; "p";

y ahora tecleamos:

PRINT PEEK 23689

obtendremos un 14 a la derecha de **p** y que representa la diferencia entre 24 y la línea donde debería ser impreso —sobre la pantalla— el siguiente carácter.

Estas dos últimas direcciones de memoria nos permiten averigurar dónde se producirá la siguiente impresión de pantalla.

Dirección 23692

Con un **POKE 23692, 255** desaparece el mensaje de **SCROLL?** y éste se produce automáticamente.

Pruebe estas líneas:

```
10 POKE 23692,255
20 FOR x=0 TO 100: PRINT "p"
30 NEXT x
```

Para finalizar, en ocasiones puede ser muy importante saber si un programa corre sobre un Spectrum 48K, para ello basta con utilizar:

PRINT PEEK 23732 + 256 * PEEK 23733

Si el programa corre sobre un Spectrum 48 K en el margen superior izquierdo de la pantalla aparecerá el número 65535.

El mapa de memoria lo podrá estudiar pormenorizadamente en la SECCION V.

En el Spectrum, cada vez que apretamos una tecla, algo sucede o se predispone a suceder. En este sentido, nuestro mejor colaborador es el CURSOR que queda representado en la pantalla por una letra parpadeante, que, a su vez, nos indica el *modo* en que el ordenador va a trabajar. Estos *modos* son:

K L G C E

Estas letras no han sido elegidas al azar, antes al contrario responden a un criterio claro y elemental: son las iniciales de las palabras inglesas que definen las funciones que cumplen.

Así, la letra **K** es la inicial de **Keyword** –o palabra clave correspondiente a cualquiera de las instrucciones del BASIC de SINCLAIR– y nos indica que el computador interpretará la próxima tecla apretada como la instrucción BASIC que figura sobre la cara de la tecla.

La letra **L** es la inicial de **letter,** previniéndonos el cursor en este modo, que interpreta el teclado como si fuera una máquina de escribir.

Tanto el modo **K** como el **L** son mostrados en pantalla automáticamente y su forma de funcionamiento es común a todo tipo de Spectrum.

Los otros modos se estudian a continuación, diferenciando el "plus" del "convencional".

La letra **G** es la inicial de **graphics** y evidentemente nos indica que el ordenador está predispuesto a trabajar en modo gráfico –o, lo que es igual, va a utilizar los caracteres gráficos–. Para entrar en este *modo,* basta con apretar la tecla **GRAPH** en el "plus" (o **CAPS SHIFT** y la tecla GRAPHICS en el "convencional") con lo cual el cursor cambia a **G.** Para salir de este *modo,* se repite el proceso.

La letra **C** es la inicial de **capitals** que significa mayúsculas. Cuando el cursor está en este modo nos indica que todas las letras serán consideradas mayúsculas. Para entrar en este *modo,* en el "plus", sólo necesitamos apretar **CAPS LOCK,** mientras que en el "convencional" debemos apretar **CAPS SHIFT** y **CAPS LOCK.** Para salir de este *modo,* repetir el proceso.

Para conseguir una sola letra en mayúsculas, apretar CAPS SHIFT y la letra deseada.

Por último, la letra **E** corresponde a la inicial de **extended,** refiriéndonos así a todos los comandos ampliados de la máquina y, para el "plus", apretamos exclusivamente EXTEND MODE, y, en el "convencional", CAPS SHIFT y SYMBOL SHIFT. De este modo se sale automáticamente cuando se aprieta, a continuación, cualquier tecla.

Para terminar estas líneas sobre el teclado, vamos a aplicar todos los modos posibles sobre una tecla, la J por ejemplo:

Above	Key	Below
BLUE EDIT	1 !	DEF FN
RED CAPS LOCK	2 @	FN
MAGENTA TRUE VIDEO	3 #	LINE
GREEN INV. VIDEO	4 $	OPEN#
CYAN ⇦	5 %	CLOSE#
YELLOW ⇩	6 &	MOVE
WHITE ⇧	7	ERASE
⇨	8 (	POINT
GRAPHICS	9)	CAT
BLACK DELETE	Ø _	FORMAT

Above	Key	Below
SIN	Q <= PLOT	ASN
COS	W <> DRAW	ACS
TAN	E >= REM	ATN
INT	R < RUN	VERIFY
RND	T > RAND	MERGE
STR $	Y AND RETURN	[
CHR $	U OR IF	]
CODE	I AT INPUT	IN
PEEK	O ; POKE	OUT
TAB	P " PRINT	©

Above	Key	Below
READ	A STOP NEW	~
RESTORE	S NOT SAVE	|
DATA	D STEP DIM	\
SGN	F TO FOR	[
ABS	G THEN GOTO	]
SOR	H ↑ GOSUB	CIRCLE
VAL	J – LOAD	VAL $
LEN	K + LIST	SCREEN $
USR	L = LET	ATTR
	ENTER	

Above	Key	Below
	CAPS SHIFT	
LN	Z : COPY	BEEP
EXP	X £ CLEAR	INK
L PRINT	C ? CONT	PAPER
L LIST	V / CLS	FLASH
BIN	B * BORDER	BRIGHT
IN KEY $	N , NEXT	OVER
PI	M . PAUSE	INVERSE
	SIMBOL SHIFT	
	BREAK SPACE	



Teclado del SPECTRUM

VAL

J – LOAD

VAL $

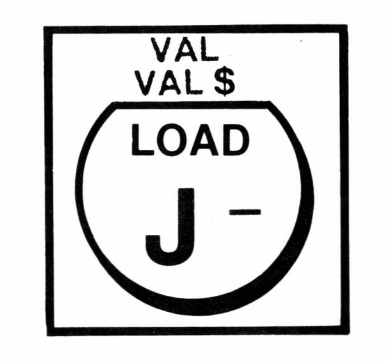
TRUE VIDEO | INV VIDEO | BLUE DEF FN ! 1 | RED FN @ 2 | MAGENTA LINE # 3 | GREEN OPEN # $ 4 | CYAN CLOSE # % 5 | YELLOW MOVE & 6 | WHITE ERASE ' 7 | POINT (8 | CAT) 9 | BLACK FORMAT _ Ø | BREAK

DELETE | GRAPH | SIN ASN PLOT Q <= | COS ACS DRAW W <> | TAN ATN REM E >= | INT VERIFY RUN R < | RND MERGE RAND T > | STR $ [RETURN AND Y | CHR $] IF OR U | CODE IN INPUT AT I | PEEK OUT POKE O | TAB © PRINT P

EXTEND MODE | EDIT | READ ~ NEW STOP A | RESTR | SAVE NOT S | DATA \ DIM STEP D | SGN { FOR TO F | ABS } GOTO THEN G | SQR CIRCLE GOSUB H ↑ | VAL VAL $ LOAD J – | LEN SCREEN $ LIST K + | USR ATTR LET L = | ENTER

CAPS SHIFT | CAPS LOCK | LN BEEP COPY Z : | EXP INK CLEAR X £ | L PRINT PAPER CONT C ? | L LIST FLASH CLS V / | BIN BRIGHT BORDER B * | IN KEY $ OVER NEXT N | PI INVERSE PAUSE M | . | CAPS SHIFT

SYMBOL SHIFT | ; | " | ⇦ | ⇨ | | ⇧ | ⇩ | , | SYMBOL SHIFT



Asumimos que al comenzar este ejercicio, estamos en modo **K** y, consiguientemente, al apretar la tecla J nos aparecerá en pantalla la palabra LOAD y, a la vez, el cursor cambia al modo **L**, con lo cual, al apretar nuevamente la tecla en cuestión, obtendremos la letra **J** en minúsculas. Si en estas condiciones se aprieta CAPS LOCK (o CAPS SHIFT y CAPS LOCK en el "convencional") el cursor pasa a modo **C**, para indicarnos que, si apretamos otra vez la J, la obtendremos en mayúsculas. Para salir del modo **C**, retornando al **L**, repetimos el proceso.

También podemos obtener la J mayúscula apretando simultáneamente esta tecla y CAPS SHIFT.

En el "plus", todos los símbolos que están inmediatamente encima, o encima y a la derecha, del símbolo principal que figura en la tecla, se obtienen apretando dicha tecla y SYMBOL SHIFT. En el caso del "convencional" este símbolo figura en rojo en la superficie de la tecla. Pruebe con SYMBOL SHIFT y J.

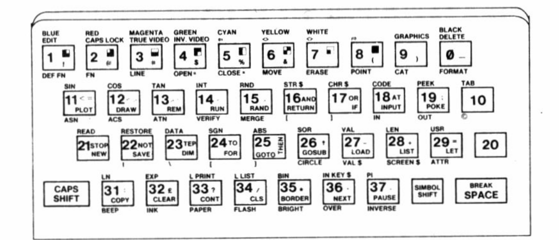
Para acceder a los símbolos y comandos situados fuera de la tecla propiamente dicha, debemos pasar al modo **E**, lo cual implica, volviendo a nuestro ejemplo, apretar **EXTEND MODE** en el "plus" o CAPS SHIFT y SYMBOL SHIFT en el "convencional" y ya, con el cursor en modo **E**, obtendremos VAL con solo apretar la **J** y VAL$ si apretamos simultáneamente la **J** y **SYMBOL SHIFT.**

El modo **G** sólo es utilizable cuando se aprieten teclas con algún carácter gráfico predefinido. Pruebe apretando **GRAPH** (o CAPS SHIFT y GRAPHICS en el "convencional"); cuando el cursor se transforme en la **G** parpadeante, apriete la letra 6, por ejemplo, con lo que aparecerá en la pantalla el símbolo gráfico que figura en la superficie de la tecla 6.

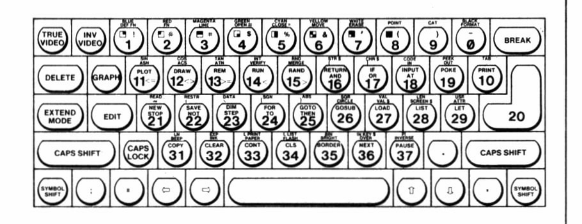
CROQUIS DE ACCESO RAPIDO A LOS COMANDOS DEL TECLADO

En la página siguiente ofrecemos un cuadro en el que por orden alfabético pueden localizarse los distintos comandos del Spectrum, tanto en su versión normal como en la versión Plus.

Croquis de acceso rápido a los comandos del teclado



K significa modo K del cursor.
E significa modo E del cursor.



Un caracter + CS significa apretar simultáneamente esa letra y CAPS SHIFT.
Un caracter + SS significa apretar simultáneamente esa letra y SYMBOL SHIFT.

Comandos por orden alfabético	Número de referencia	Controlado por la(s) tecla(s)	En modo
ABS	26	G	E
ACS	12	W+SS	E
AND	16	Y+SS	K
ASN	11	Q+SS	E
AT	18	I+SS	K
ATN	13	E+SS	E
ATTR	29	L+SS	E
BEEP	31	Z+SS	E
BIN	35	B	E
BORDER	35	B	K
BRIGHT	35	B+SS	E
CAT	9	9+SS	E
CHR$	17	U	E
CIRCLE	26	H+SS	E
CLEAR	32	X	K
CLOSE #	5	5+SS	E
CLS	34	V	K
CODE	18	I	E
CONT	33	C	K
COPY	31	Z	K
COS	12	W	E
DATA	23	D	E
DEF FN	1	1+SS	E
DELETE	Ø	Ø+CS	
DIM	23	D	K
DRAW	12	W	K
EDIT	1	1+CS	
ERASE	7	7+SS	E
EXP	32	X	E
FLAGH	34	V+SS	E
FN	2	2+SS	E
FOR	24	F	K
FORMAT	Ø	Ø+SS	E
GOSUB	26	H	K
GOTO	25	G	K
IF	17	U	K
IN	18	I+SS	E
INK	32	X+SS	E
INKEY$	36	N	E
INPUT	18	I	K
INT	14	R	E
INVERSE	37	M+SS	E
LEN	28	K	E
LET	29	L	K
LINE	3	3+SS	E

Comandos por orden alfabético	Número de referencia	Controlado por la(s) tecla(s)	En modo
LIST	28	K	K
LLIST	34	V	E
LN	31	Z	E
LOAD	27	J	K
LPRINT	33	C	E
MERGE	15	T+SS	E
MOVE	6	6+SS	E
NEW	21	A	K
NEXT	36	N	K
NOT	22	S+SS	K
OPEN #	4	4+SS	E
OR	17	U+SS	K
OUT	19	O+SS	E
OVER	36	N+SS	E
PAPER	33	C+SS	E
PAUSE	37	M	K
PEEK	19	O	E
PI	37	M	E
PLOT	11	Q	K
POINT	8	8+SS	E
POKE	19	O	K
PRINT	10	P	K
RAND	15	T	K
READ	21	A	E
REM	13	E	K
RESTORE	22	S	E
RETURN	16	Y	K
RND	15	T	E
RUN	14	R	K
SAVE	22	S	K
SCREEN$	28	K+SS	E
SGN	24	F	E
SIN	11	Q	E
SQR	26	H	E
STEP	23	D+SS	K
STOP	21	A+SS	K
STR$	16	Y	E
TAB	10	P	E
TAN	13	E	E
THEN	25	G+SS	K
TO	24	F+SS	K
USR	29	L	E
VAL	27	J	E
VAL$	27	J+SS	E
VERIFY	14	R+SS	E

DIAGRAMAS DE FLUJO

Programar es una actividad lógica que requiere claridad y precisión en todas sus etapas. De entre ellas, la más importante es la que se refiere al planteamiento del propio programa.

Definir los objetivos a lograr y hacerlo de forma certera ahorra esfuerzos inútiles. Para ello, debe plantearse la siguiente pregunta:

¿Que se pretende conseguir con el programa?

La respuesta a esta cuestión no siempre es fácil y, a mayor complejidad, necesitará más tiempo de reflexión.

Una vez superada esta fase, aparece indefectiblemente otro interrogante:

¿Cómo organizar el programa?

Es claro que una forma de organizar un programa consiste en sentarse ante el computador y comenzar a teclear de manera que, finalmente, el listado *obligue* a la máquina a dar una respuesta, aunque no se haya dado una estructura lógica al trabajo.

En este APENDICE, vamos a tratar **los diagramas de flujo** u **organigramas**, gracias a los cuales se obtiene un boceto, a grandes rasgos, de lo que será el programa.

La mayor ventaja que se deriva de la realización de *los diagramas de flujo* es el esfuerzo de síntesis al que se somete el programador, con la consiguiente claridad de ideas que de esto se deriva.

Un diagrama de flujo consiste, en definitiva, en una serie de **símbolos estándar**, unidos por **flechas** para indicar los caminos que la secuencia del programa puede tomar.

Cada *símbolo estándar* implica, por su forma, una determinada actividad, pudiéndose escribir dentro de ellos todo lo que sirva para darlos mayor claridad.

Los símbolos estándar generalmente utilizados en *los diagramas de flujo* son:

Con **el rectángulo** se expresa la idea de acción, que será llevada a cabo por un comando BASIC.

Con **el rombo** se exige la toma de decisión entre dos posibilidades.

Con **este paralelogramo** se indica que entran datos en el computador o salen de él.

Con **el círculo** se conectan las partes del programa que no lo están por medio de flechas, para lo que basta escribir en su interior la referencia oportuna.

Con **esta figura compuesta** se representan el principio y el final de determinadas secuencias, ya sean programas completos o subrutinas.

Antes de pasar a unos ejemplos que clarifiquen lo expuesto hasta aquí, quisiéramos hacer hincapié en que **los organigramas no son consecuencia de los programas, sino que la estructura de cada programa es una consecuencia del organigrama correspondiente.**

Los *diagramas* que figuran a continuación deberán ser hechos a mano por el programador a fin de sacar al lector de cualquier rigor academicista.

Los diagramas y sus símbolos están pensados para ser usados libremente por el programador de tal forma que se conviertan en un buen útil de trabajo que pueda manejarse sin ninguna rectricción.

Dicho esto, pasemos a analizar algunos problemas elementales susceptibles de ser convertidos en programas.

EJEMPLO:

1. Desarrollar un programa que permita sumar dos números introducidos por teclado, en una secuencia ininterrumpida, y se impriman los resultados en pantalla.

DIAGRAMA:

PRINCIPIO

Entra un número por teclado

Entra otro número por teclado

Se suman los dos números anteriores

Imprimir el resultado en pantalla

¿Otra suma?

SI

NO

FIN

PROGRAMA:

```
5 REM Sumar dos números cualesquiera

1Ø INPUT A

2Ø INPUT B

3Ø LET C = A + B

4Ø PRINT C

5Ø INPUT "Otra suma?"; S$

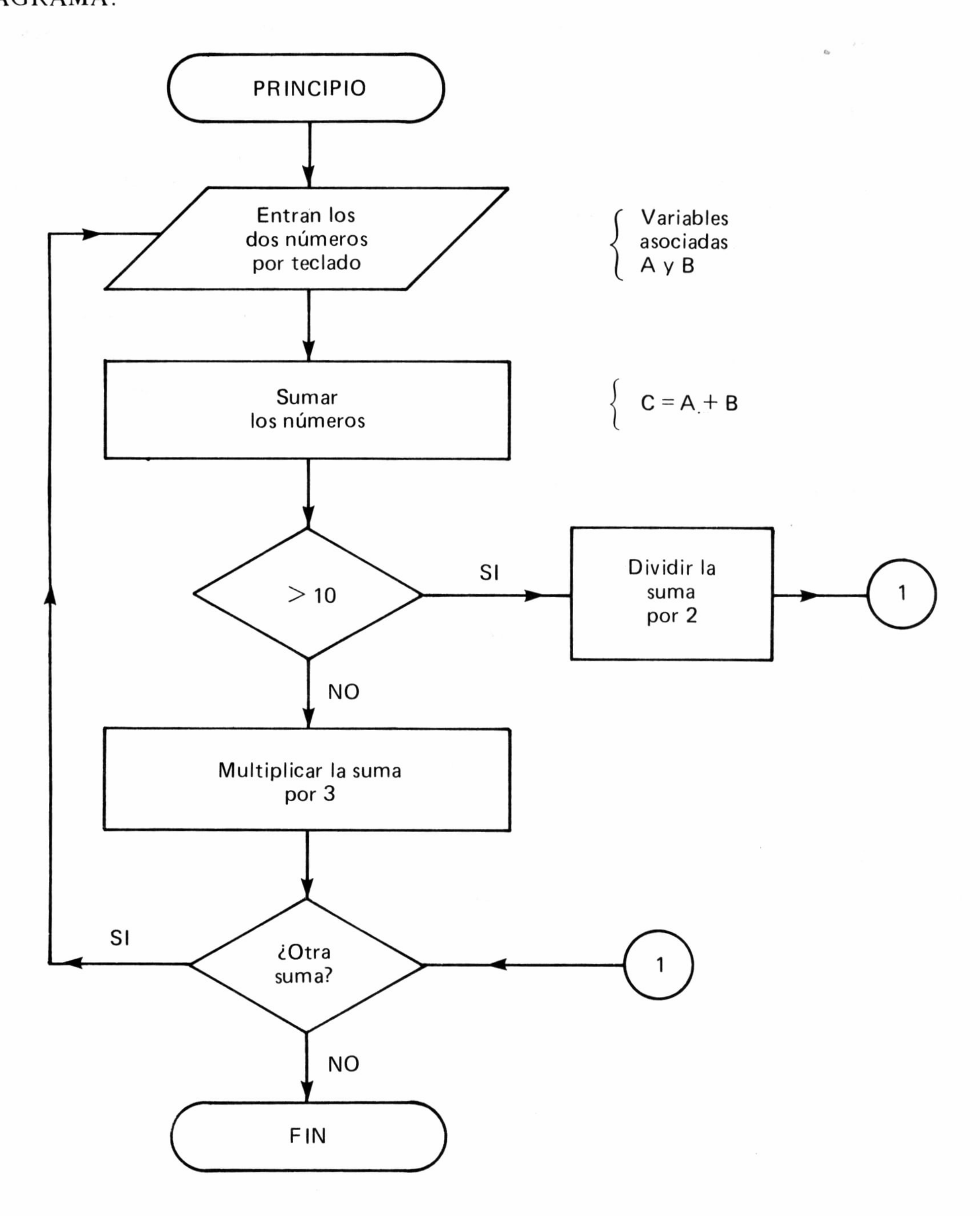
6Ø IF S$ = "S" THEN GO TO 1Ø

7Ø STOP
```

EJEMPLO:

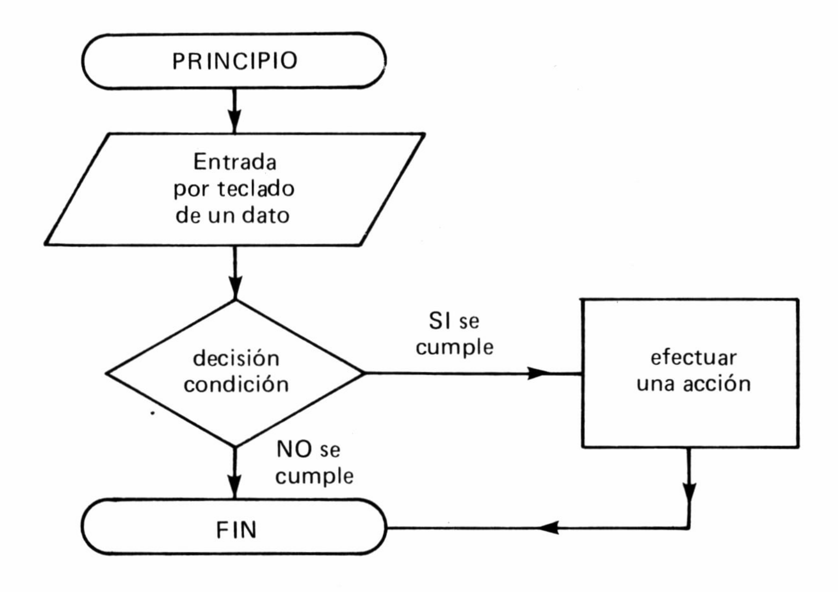
2. Desarrollar un programa que, al ejecutarlo, permita introducir dos números por teclado y si su suma es mayor que diez, esta se divida por dos; en otro caso, que se multiplique por 3.

DIAGRAMA:



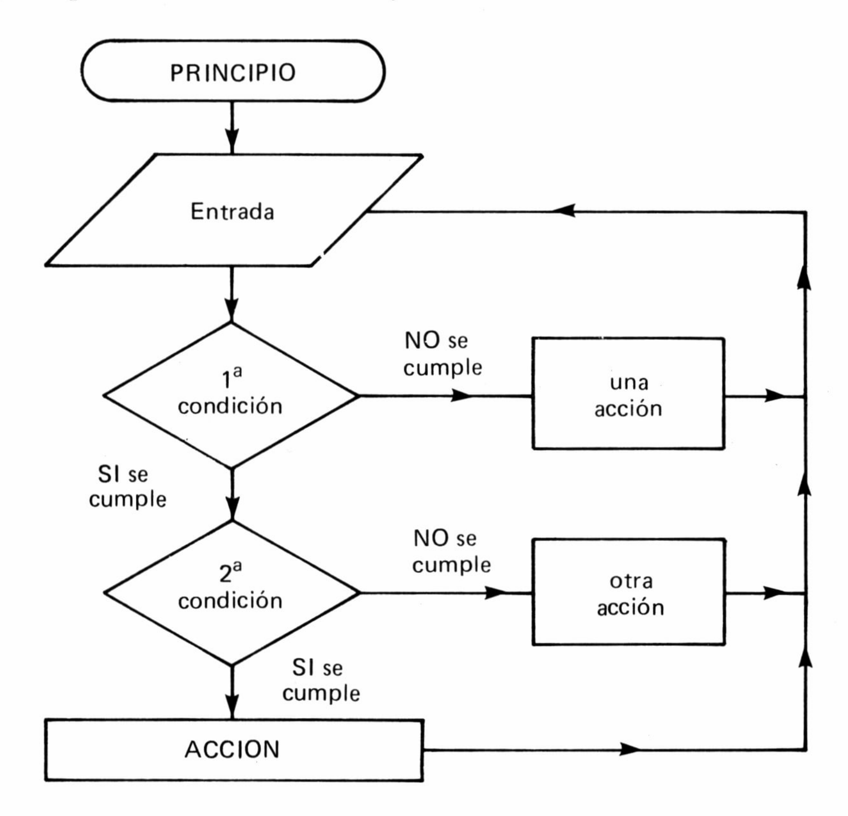
EJEMPLO:

3. Diagrama para una decisión sencilla:



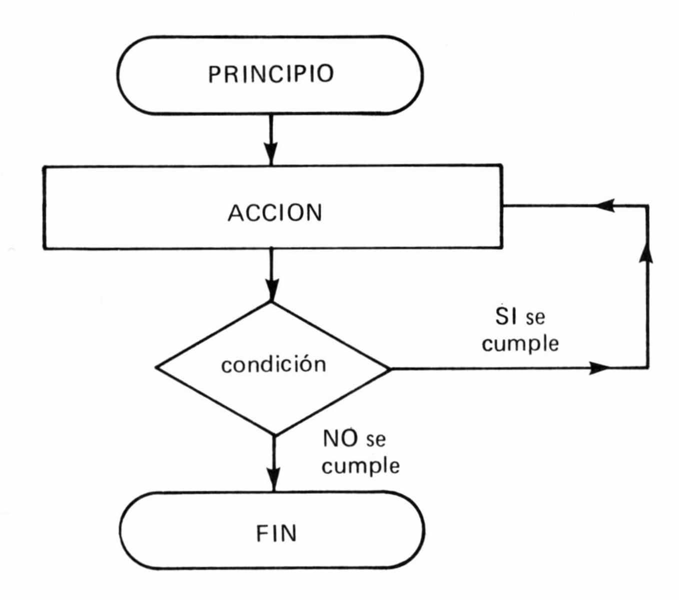
EJEMPLO:

4. Diagrama para más de una decisión y bucle:



EJEMPLO:

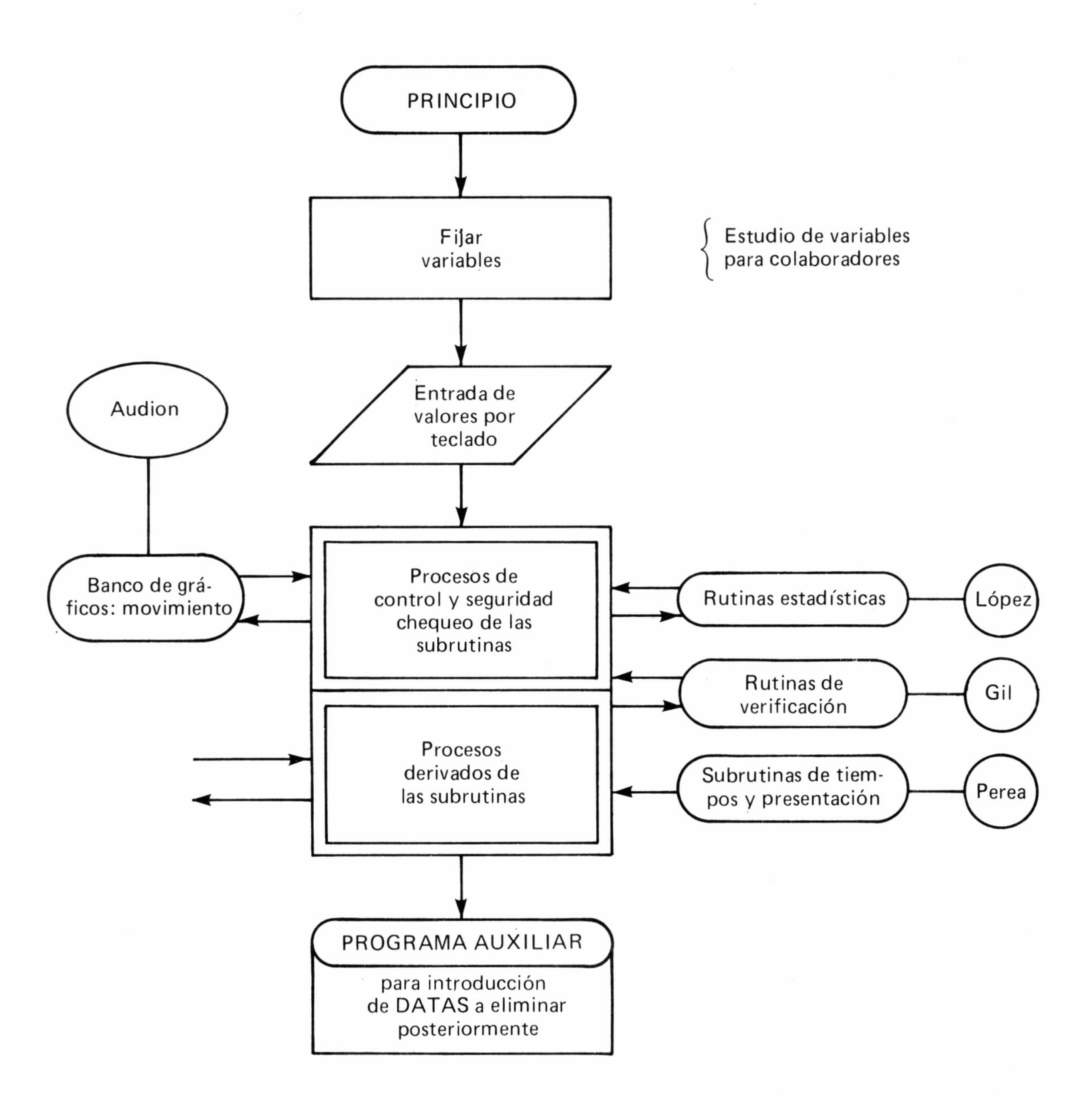
5. Diagrama para un bucle indefinido hasta que no se cumpla una condición:



A medida que los conocimientos de BASIC del programador aumentan, su capacidad para resolver problemas se hace paulatinamente mayor y, en consecuencia, los diagramas tendrán que plantearse desde perspectivas que abarquen mayores generalidades del problema a resolver, del cual se sacarán, a su vez, unos diagramas más precisos.

EJEMPLO:

6. Posible diagrama para un programa de pruebas objetivas:



Todos estos bloques son en sí mismos un programa diferente cada uno, por lo que algunos de ellos los van a realizar los programadores cuyos nombres figuran anejos a los mismos, con el fin de hacer más fácil la tarea.

Cada uno de estos colaboradores debe presentar su propio diagrama al coordinador del proyecto para poder estudiar conjuntamente los problemas que se presentan y las incompatibilidades que puedan surgir. Esta suele ser la forma de trabajar en programas complejos.

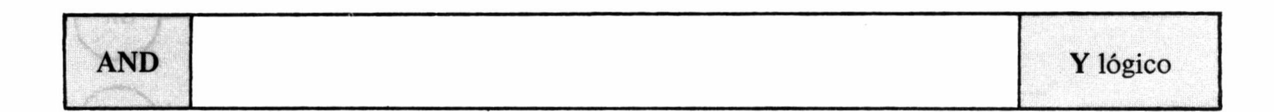
Apéndice C FUNCIONES LOGICAS APLICADAS

Su aplicación a la intrucción IF/THEN

Al hablar sobre **los operadores de relación**, dijimos que estos sólo actúan sobre dos operandos cuyo resultado únicamente puede ser *cierto* o *falso*. Por ejemplo, en **A>B** sólo hay dos posibilidades esperadas: **A** es mayor que **B** o no lo es.

Los operadores lógicos actúan entre *operandos* conformados por *operadores de relación*, lo cual implica que *los operadores lógicos* sólo actúan entre dos posibilidades: *cierto* y *falso*.

En este Apéndice vamos a estudiar la aplicación de **los operadores lógicos AND, OR** y **NOT** en combinación con la instrucción **IF/THEN.**



Analicemos desde un punto de vista estrictamente lógico la siguiente expresión:

IF (A = 15) AND (B > 100) THEN PRINT "bien"

Las posibilidades que se nos ofrecen son que **A** sea o no igual a **15** y que **B** sea o no mayor que **100.** En función de ambas alternativas, escribiremos o no la palabra **"bien".**

En el ejemplo tenemos dos operadores de relación, "=" y ">", y el resultado de aplicarlos como sabemos entre dos operandos, sólo puede ser *cierto* o *falso*. Asociemos *cierto* con el símbolo **1** y *falso* con el **Ø.**

De aquí deducimos que, las combinaciones posibles de **1** y **Ø** que nos ofrecen ambas *operaciones de relación* son:

A = B ⇒ cierto ⇒ 1
otra posibilidad ⇒ falso ⇒ Ø
B > 1ØØ ⇒ cierto ⇒ 1
otra posibilidad ⇒ falso ⇒ Ø

NOTA: El símbolo ⇒ significa "implica".

Volviendo a la expresión inicial, escribiremos las posibilidades de que la palabra "bien" sea escrita:

Si **A** = **B** (cierto ó 1) **Y** **B** > **100** (cierto ó 1) entonces **sí** se escribe
Si (falso ó ∅) **Y** (falso ó ∅) entonces **no** se escribe
Si (falso ó ∅) **Y** (cierto ó 1) entonces **no** se escribe
Si (cierto ó 1) **Y** (falso ó ∅) entonces **no** se escribe

Es decir que el operador **lógico y** exige que los dos operadores sobre los que actúa sean 1 –*cierto*– para permitir que suceda la acción que le sigue.

Para concluir, hagamos igual a **x** el resultado (∅ ó 1) de la primera operación de relación (**A** = **15**, en el ejemplo) e igual a **y** el resultado (∅ ó 1) de la segunda operación de relación (**B** > **100**, en el ejemplo); pudiendo presentar las posibilidades que ofrece el operador **lógico Y** (AND, en inglés) mediante la siguiente

TABLA DE LA VERDAD: AND

x	y	x AND y	COMENTARIOS
1	1	1	Se permite la acción subsecuente
∅	∅	∅	Se prohíbe la acción subsecuente
∅	1	∅	Se prohíbe la acción subsecuente
1	∅	∅	Se prohíbe la acción subsecuente

En lo sucesivo nos referiremos directamente a la *tabla de la verdad* de cada operador lógico, evitando la explicación que nos lleva a ella.

EJEMPLO:

Para comenzar los estudios de Bachillerato es necesario tener 14 años y haber obtenido una calificación mínima de 5 puntos en la E.G. Estudie este programa y ejecútelo.

PROGRAMA

```
10 INPUT "Algún alumno?"; A$
20 IF A$ = "sí" THEN GO SUB 60
30 IF A$ = "no" THEN STOP
40 RETURN
50 CLS
60 INPUT "Nombre del alumno?; N$: PRINT N$
70 INPUT "Edad?"; E: PRINT E
80 INPUT "Calificación?"; C: PRINT C
90 IF (E>14) AND (C<5) THEN GO TO 140
100 PRINT "Admitido"
110 PRINT
120 INPUT "Otro alumno?"; A$
130 GO TO 20
140 PRINT "Rechazado": GO TO 110
```

Este *operador lógico* permite la acción subsecuente siempre que uno cualquiera de los operadores de relación sea **1** (cierto).

Volviendo al ejemplo del operador anterior, pero cambiando éste por el actual (**OR**), tendremos en términos de BASIC:

IF (A = 15) OR (B > 100) THEN PRINT "bien"

Según la siguiente

TABLA DE LA VERDAD: OR

x	y	x OR y	COMENTARIOS
1	1	1	Se permite la acción subsecuente
Ø	Ø	Ø	No se permite la acción subsecuente
Ø	1	1	Se permite la acción subsecuente
1	Ø	1	Se permite la acción subsecuente

Podemos deducir que la palabra "bien" será impresa en pantalla en el caso de que **A** sea igual a **15 O B** sea mayor que **1ØØ**.

EJEMPLO:

¿Recuerda el programa de Miss Universo? Compárelo con éste, en el que se ha introducido OR. Ejecútelo y observe los resultados.

PROGRAMA

```
 1Ø INPUT "Alguna candidata:"; O$
 2Ø IF O$ = "sí" THEN GO SUB 6Ø
 3Ø IF O$ = "no" THEN STOP
 4Ø RETURN
 5Ø CLS
 6Ø INPUT "Nombre?"; N$: PRINT N$
 7Ø INPUT "Talla"; T: PRINT T
 8Ø INPUT "Peso?"; P: PRINT P
 9Ø IF (T < 1.7) OR (P > 6Ø) THEN GO TO 14Ø
1ØØ PRINT "Admitida"
11Ø PRINT
12Ø INPUT "Otra candidata?"; O$
13Ø GO TO 2Ø
14Ø PRINT "Rechazada": GO TO 11Ø
```

Este es el más sencillo de *los operadores lógicos,* ya que el resultado es falso (Ø) cuando el operador de relación es cierto (1), y cierto (1) cuando el operador de relación es falso (Ø), según la siguiente

TABLA DE LA VERDAD: NOT

x	NOT x	COMENTARIOS
1	Ø	No se permite la acción subsecuente
Ø	1	Sí se permite la acción subsecuente

EJEMPLO:

IF NOT (A = B) THEN PRINT "bien"

Esto implica que la palabra "bien" será impresa en pantalla sólo en el caso de que A no sea igual a B.

Apéndice D — TROCEADO DE CADENAS

Por **cadena**, y en términos de programación, entendemos cualquier concatenación de caracteres.

Entre otras cosas, hay que tener en cuenta que una cadena sólo se puede igualar a una variable de caracteres. Tal sería el caso de

A$ = "ROMARBELLAVE, es una ? > 1"

En este ejemplo la cadena está compuesta por letras mayúsculas y minúsculas, dos signos de putuación, un espacio y un par de símbolos matemáticos. En definitiva, caracteres en general.

Trocear una cadena significa subdividirla en cadenas menores. A este tema dedicaremos el Apéndice presente.

Sobre la variable **A$** fijada más arriba, aplicaremos los comandos oportunos.

Este sistema lo seguiremos con todas las instrucciones de este apéndice.

Entendemos que, como introducción a esta materia, son suficientes los COMENTARIOS para hacerse idea de la aplicación de cada comando.

Posibilidad 1

Extraer, de la cadena representada por **A$**, una subcadena con los cuatro caracteres situados más a la izquierda.

Ejemplo	COMENTARIOS
5Ø LET B$ = A$ (1 TO 4) **6Ø PRINT B$**	La ejecución de estas dos líneas de programa obligará la impresión en pantalla de la cadena ROMA.

Posibilidad 2

Extraer una subcadena de **A$** con los cinco caracteres situados más a la derecha.

Ejemplo	COMENTARIOS
5Ø LET B$ = A$ (2ØTO 24) **6Ø PRINT B$**	La ejecución de esta dos líneas nos dará en pantalla: **a ? > 1** Fíjese el lector que el primer carácter de esta subcadena es un espacio. Idéntico resultado.

Posibilidad 3

Extraer una subcadena de A$ que comience en su tercer carácter y tenga una longitud de 8 caracteres.

Ejemplo	COMENTARIOS
5Ø LET B$ = A$ (3 TO 1Ø) **6Ø PRINT B$**	Al ejecutar estas líneas obtendremos: **MARBELLA**

Posibilidad 4

Extraer un solo carácter de la cadena A$.

Ejemplo	COMENTARIOS
50 LET B$ = A$ (6 TO 6) **60 PRINT B$**	En pantalla aparecerá **B**.

Todo lo visto más arriba y algunas otras formas de aplicar el comando TO con idéntico resultado, lo podemos ver en el siguiente ejemplo:

Programa	**Comentarios**
10 LET A$= "ROMARBELLAVE"	La variable A$ es valorada.
20 PRINT A$ (TO 4)	Imprimo todos los caracteres desde el principio hasta el cuarto, de la A$.
30 PRINT A$ (3 TO 10)	Imprimo todos los caracteres desde el tercero al décimo, ambos inclusive, de la A$.
40 PRINT A$ (8 TO)	Imprimo todos los caracteres del octavo al final de la A$.
50 PRINT A$ (6 TO 6)	Imprimo el sexto caracter únicamente.
60 PRINT A$ (1 TO 12)	Imprimo la cadena del principio al final. Igual resultado obtendrías con PRINT A$.
70 LET B$ = A$ (6 TO 10)	Dejamos la variable B$ igual a la subcadena A$ indicada.

ENCICLOPEDIA DEL SPECTRUM

En esta enciclopedia se estudian la práctica totalidad de los temas que el usuario del Spectrum necesita para dominar su máquina, todos ellos tratados de una forma coloquial y agradable.

Este esfuerzo editorial está basado en la calidad didáctica, ya contrastada, de cada una de las secciones que lo componen, que ya han sido tratadas en títulos independientes que separadamente han logrado una aceptación extraordinaria:

- Cómo programar su Spectrum
- Cómo usar los colores y los gráficos en el Spectrum
- Basic para maestros

La revista "Muy interesante" ha escrito, refiriéndose al autor:

"Más claro, el agua". "Muy ameno por el tratamiento coloquial del texto".

Y la revista "ZX" comenta:

"Con la frase —Hola, soy tu Spectrum— se inicia una larga charla en la que nuestro querido Spectrum es el profesor".

En resumen, con el *"Spectrum: Plus Ultra"* los aficionados a la microinformática podrán iniciarse en aspectos que van desde la programación en BASIC, hasta el manejo del Microdrive, pasando por el Código Máquina.

Un fichero de comandos BASIC, un croquis de acceso rápido a las teclas y un detallado mapa de Memoria hacen de esta enciclopedia el manual imprescindible del usuario del Spectrum.



Magallanes, 25 - 28015 Madrid

ISBN: 84-283-1391-1